Clube Delphi

Quick Update
Injeção de Dependência

Easy Delphi
Tratamento de Exceções

Mini-Curso
Padrões de Projeto - Parte 2

Ano 9 :: Edição 123 :: R$ 11,90

DevMedia

ISSN 1517990-7

Delphi com estilo

Usando CSS no Prism

Brinde

Comprando esta edição ganhe 3 vídeo-aulas:

• Entendendo mensagens do Windows • Interceptando mensagens do Windows • Disparando mensagens

Lazarus e FreePascal
Conheça o IDE Lazarus e o compilador FreePascal, uma ótima alternativa Open Source – Parte 2

Delphi Prism e ASP.NET
Reaproveite seus conhecimentos Win32 / Desktop para criar aplicações Web – Parte 6

Dificuldades em encontrar conteúdo sobre .Net para iniciantes?

A DevMedia simplifica pra você!

easy .net magazine

Easy .Net Magazine:

A primeira revista digital feita especialmente para quem está dando os primeiros passos em .Net!

Você encontrará exemplos e modelos fáceis e práticos, noções introdutórias e conceitos básicos.

Conheça as vantagens:

- Vídeos e apresentações "embutidos" nas páginas;
- Ilustrações, diagramas e modelos apresentados em PowerPoint;
- Entrevistas em vídeo;
- Códigos-fonte que podem ser copiados com CTRL+C/CTRL+V.

A Easy.Net não é feita em PDF ou Flash!
Ela utiliza um aplicativo 100% online.

DevMedia group

Acesse agora a edição Nº 1 totalmente grátis e comprove

www.devmedia.com.br/easynet

Easy.Net Magazine, seu curso inicial de .Net!

Sumário

Quick Update

06 – Injeção de Dependência
[*Paulo Quicoli*]

Delphi

Web

08 – Delphi com estilo
Usando CSS no Prism
[*Daniel Sobrinho Laporte*]

Tutorial

18 – Lazarus e FreePascal
Conheça o IDE Lazarus e o compilador FreePascal, uma ótima alternativa Open Source – Parte 2
[*Daniel Simões Almeida*]

Boas Práticas

Boas Práticas

28 – Design Patterns - Parte 2
Abstract Factory e Factory Methods
[*Vitor Luiz Rubio*]

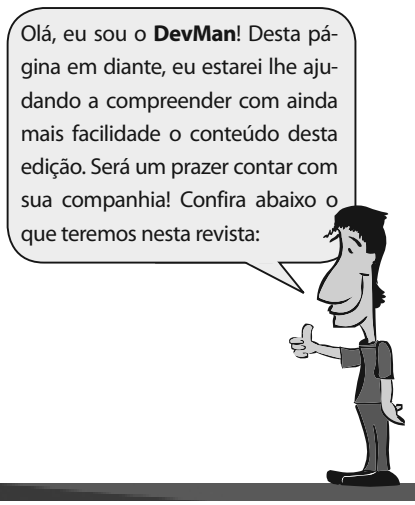
Easy Delphi

Easy Delphi - Mini-Curso – Tutorial – Web – Novidades

40 – Delphi Prism e ASP.NET
Reaproveite seus conhecimentos Win32 / Desktop para criar aplicações Web – Parte 6
[*Guinther Pauli*]

Easy Delphi

44 – Entendendo e tratando as exceções no Delphi
Desenvolva aplicações robustas, eficientes e tolerantes a falhas
[*Rodrigo Carreiro Mourão*]



Brinde na web desta edição - 3 vídeos! Confira:

3 Vídeos

• **Entendendo mensagens do Windows -** *http://www.devmedia.com.br/post-17828-Mensagens-do-Sistema-Operacional--Curso-Online-Dominando-a-Orientacao-a-Objetos-e-Componentes-em-Delphi--Avancado-Parte-29.html*

• **Interceptando mensagens do Windows -** *http://www.devmedia.com.br/post-17829-Interceptando-Mensagem--Curso-Online-Dominando-a-Orientacao-a-Objetos-e-Componentes-em-Delphi--Avancado-Parte-30.html*

• **Disparando mensagens -** *http://www.devmedia.com.br/post-17830-Alterando-comportamento--TWinControl-Curso-Online-Dominando-a-Orientacao-a-Objetos-e-Componentes-em-Delphi--Avancado-Parte-31.html*

Para acessar os vídeos utilize os seguintes dados
Login: ***DVM.CD*** *Senha:* ***C0RB1N***

Gostou das vídeo aulas? O portal *www.devmedia.com.br* possui mais de **2 mil vídeo aulas** e **dezenas de cursos online** sobre desenvolvimento de software! Agora você pode comprar as vídeo aulas que preferir e fazer sua própria combinação de vídeos! Saiba mais em *www.devmedia.com.br/creditos*

Dê seu feedback sobre esta edição!

A ClubeDelphi tem que ser feita ao seu gosto. Para isso, precisamos saber o que você, leitor, acha da revista!

Dê seu voto sobre esta edição, artigo por artigo, através do link:
www.devmedia.com.br/clubedelphi/feedback

Para votar, você vai precisar do código e senha de banca desta edição, que é:
Login: **DVM.CD** Senha: **C0RB1N**

Clube Delphi

Ano 9 - 123ª Edição 2010 - ISSN 1517990-7 Impresso no Brasil

Corpo Editorial

Editor Geral
Guinther Pauli
guinther@devmedia.com.br

Editor Técnico
Paulo Quicoli
pauloquicoli@devmedia.com.br

Daniel Laporte
daniel.sobrinho@devmedia.com.br

Rodrigo Carreiro Mourão
rodrigocarreiro@rmfactory.com.br

Equipe Editorial
Vitor Luiz Rubio
vitorrubio@gmail.com

Rafael Stavarengo
faelsta@gmail.com

Capa e Diagramação
Romulo Araujo
romulo@devmedia.com.br

Coordenação Geral
Daniella Costa - daniella@devmedia.com.br

Revisão e Supervisão
Thiago Vincenzo - thiago.v.ciancio@devmedia.com.br

Distribuição
Fernando Chinaglia Dist. S/A
Rua Teodoro da Silva, 907
Grajaú - RJ - 206563-900

Atendimento ao Leitor

A DevMedia conta com um departamento exclusivo para o atendimento ao leitor. Se você tiver algum problema no recebimento do seu exemplar ou precisar de algum esclarecimento sobre assinaturas, exemplares anteriores, endereço de bancas de jornal, entre outros, entre em contato com:

Cristiany Queiroz – Atendimento ao Leitor
www.devmedia.com.br/mancad
(21) 3382-5038

Kaline Dolabella – Gerente de Marketing e Atendimento
kalined@terra.com.br
(21) 3382-5038

Publicidade

Para informações sobre veiculação de anúncio na revista ou no site e para fechar parcerias ou ações específicas de marketing com a DevMedia, entre em contato com:

Kaline Dolabella
publicidade@devmedia.com.br

Fale com o Editor!

É muito importante para a equipe saber o que você está achando da revista: que tipo de artigo você gostaria de ler, que artigo você mais gostou e qual artigo você menos gostou. Fique a vontade para entrar em contato com os editores e dar a sua sugestão!
Se você estiver interessado em publicar um artigo na revista ou no site ClubeDelphi, entre em contato com os editores, informando o título e mini-resumo do tema que você gostaria de publicar:

Guinther Pauli - Editor da Revista
guinther@devmedia.com.br

EDITORIAL

Uma das principais dificuldades dos desenvolvedores Desktop ao iniciarem o desenvolvimento para Web diz respeito ao layout de páginas. Na VCL, nossos TForms e descendentes utilizam o que chamamos de posicionamento absoluto para controles. Na Web temos diferentes formas para organizar elementos em tela, como o uso de tabelas HTML ou CSS, usado também para definir estilos para elementos. Na VCL, quando precisamos que determinado conjunto de componentes tenham características semelhantes, como por exemplo, todos os botões de OK terem a mesma aparência, temos várias saídas: podemos criar uma classe especializada, que sobrescreva propriedades no construtor, podemos criar templates de controles (recurso introduzido no Delphi 3), ou utilizar bibliotecas de Skins. Outra alternativa seria usar RTTI para procurar por classes em específico e alterar propriedades em runtime. Felizmente, no ASP.NET com Delphi Prism temos várias alternativas para centralizar a configuração de layout, como o uso de CSS e Themes / Skins (recurso nativo do ASP.NET, introduzido na versão 2.0). Além disso, com o advento da nova geração de aplicações para Web, com a forte ascensão de padrões como JavaScript, HTML5 e CSS3, dominar o uso de estilos se torna obrigatório. De fato, aplicações Web são hoje tão ricas quanto aplicações Desktop, rodam em browsers e fazem uso de computação nas nuvens, muito devido ao uso intenso desses padrões, que enriquecem a interatividade no cliente (browser). Daí o termo RIA – Rich Internet Application, que representa a nova onda do momento na criação de aplicativos para Internet. É o fim do Desktop, como apostou Steve Jobs em sua polêmica carta aberta (http://bit.ly/cauUjJ) criticando a Adobe? Não podemos afirmar, mas ficar preparado é muito oportuno. Esta edição destaca o uso de CSS com Delphi Prism, na matéria do Daniel Laporte.

Na edição passada trouxemos com exclusividade uma matéria do Daniel Simões que apresentou à comunidade Delphi o Lazarus e Free Pascal, ferramentas livre e open source para o desenvolvimento de aplicações para várias plataformas como Windows, Linux, WinCE e MacOS. Este é um projeto desenvolvido com a colaboração de vários renomados experts na área de desenvolvimento Delphi, que garantem um nível de maturidade e seriedade muito grande ao produto. Nesta edição, temos a segunda parte do artigo, onde o autor apresenta excelentes pacotes de componentes e recursos para personalizar o ambiente de desenvolvimento, incluindo famosas suítes de componentes, para relatórios e acesso a dados.

Na sessão Boas Práticas, a segunda parte do artigo do Vitor Rubio, que apresenta um dos padrões de projeto mais conhecidos, do grupo de patterns criacionais, as fábricas de objetos (padrões como Abstract Factory e Factory Method). Nesta edição o autor detalha como criar fábricas abstratas, otimizando ainda o uso do padrão com outro poderosíssimo recurso que está crescendo cada vez mais, agora também na linguagem Delphi Win32: Generics.

Na sessão Easy, continuo a nossa série que está mostrando como você pode desenvolver aplicações para Web com ASP.NET de forma muito semelhante ao que está acostumado a fazer em Delphi Win32. Na matéria do Rodrigo Carreiro, uma excelente abordagem sobre o tratamento de exceções.

Boa leitura!

Guinther Pauli

guinther@devmedia.com.br

Certificado Delphi 3,5,6,7,2005,2006,Web,Kylix

Microsoft Certified: MCP,MCAD,MCSD.NET,MCTS,MCPD (ASP.NET,C#)

guintherpauli.blogspot.com - twitter.com/guintherpauli

ClubeDelphi PLUS

A revista ClubeDelphi é parte integrante da assinatura ClubeDelphi PLUS.
Para mais informações sobre o pacote PLUS, acesse:
http://www.devmedia.com.br/clubedelphi/portal.asp

DevMedia

Sistema de créditos DevMedia

Conteúdo **completo** ao seu alcance

Com o sistema de créditos DevMedia você pode comprar individualmente mais de 4.000 vídeo aulas, 8000 artigos, além de palestras e vídeos com formações completas.

TODAS as vídeos que você sempre quis ver a partir de

R$ 0,99

ECONOMIA: você não precisa assinar 8 produtos diferentes para ter acesso ao conteúdo completo do site!

Saiba mais sobre o Sistema de Créditos!

www.devmedia.com.br/creditos

DevMedia

Quick Update: Injeção de Dependência

Paulo Quicoli

pauloquicoli@gmail.com

Editor técnico das revistas ClubeDelphi, .NET Magazine e WebMobile Magazine. Formado em processamento de dados pela FATEC-TQ. Atua como analista e desenvolvedor da ControlM Informática (www.controlm.com.br) e professor de linguagem de programação do curso técnico em processamento de dados do SENAC/Jaboticabal. Utiliza Delphi desde sua primeira versão, entusiasta do desenvolvimento orientado a objetos, tem publicado vários artigos sobre o assunto. Blog: www.pauloquicoli.spaces.live.com

Injeção de dependência é um padrão de desenvolvimento utilizado na orientação a objetos quando queremos manter baixo o nível de acoplamento entre diferentes partes de uma aplicação. Não esclareceu muito a questão, não é? A melhor forma de entender este tipo de conceito é na prática, e é isso que veremos a seguir.

Pensemos num exemplo bem simples, onde temos um site onde os clientes realizam pedidos, e ao final é emitida uma cobrança. Vamos imaginar que num primeiro momento os pagamentos serão realizados exclusivamente através de Boletos Bancários. Comecemos então criando uma classe que será responsável por esse tipo de pagamento, veja na **Listagem 1**.

Agora vamos criar a classe que irá representar os pedidos realizados. Essa classe deverá ter o método FecharPedido, que aqui neste modesto exemplo, irá simplesmente emitir um Boleto. Confira na **Listagem 2**.

Basta chamar o método FecharPedido da classe TPedido para que o boleto de cobrança seja emitido, veja como fica a chamada na **Listagem 3**. Não precisamos nem mostrar aqui o resultado que esse código irá exibir.

Isso funciona muito bem enquanto nosso site estiver trabalhando com boleto bancário. Mas e se quisermos implementar outras formas de pagamento? Como você pode notar na **Listagem 2**, o boleto é criado dentro da classe TPedido, e isso significa que toda vez que quisermos implementar uma nova forma de pagamento, teremos que alterar essa classe.

Quando temos este tipo de dependência, dizemos que a classe TPedido está fortemente acoplada à classe TBoleto. Quando temos classes fortemente acopladas, elas não podem ser substituídas sem que suas implementações sejam alteradas.

Na Injeção de Dependência utilizamos interfaces para diminuir esse acoplamento entre as classes. Vamos criar uma interface para representar as formas de pagamento que o site irá oferecer. Veja na **Listagem 4**.

E portanto, agora nossa classe TBoleto deverá implementar a interface IFormaPagamento, veja na **Listagem 5**.

Nosso pedido agora não precisa mais ter uma dependência direta à classe TBoleto, podemos criar um objeto mais genérico de cobrança, através da interface que criamos, veja na **Listagem 6**.

Note que além de termos um objeto genérico que irá representar a forma de pagamento, não estamos mais instanciando o Boleto no construtor do Pedido. O Boleto, ou qualquer outra forma de pagamento, será passado ao construtor. Dessa forma, a "dependência" entre as classes TPedido e TBoleto, é "injetada" no construtor da classe TPedido.

Isso nos permite criar outras formas de pagamento, sem termos que mexer na classe TPedido, veja por exemplo, como ficaria a classe de TCartaoCredito na **Listagem 7**.

Listagem 1. Classe TBoleto

```
TBoleto = class
public
 procedure Cobrar(Valor: Currency);
end;
procedure TBoleto.Cobrar(Valor: Currency);
begin
 Writeln('Emitir boleto no valor de ' +
   FloatToStrF(Valor,ffCurrency,15,2));
end;
```

Listagem 2. Classe TPedido

```
TPedido = class
private
 FBoleto: TBoleto;
public
 constructor Create;
 procedure FecharPedido(Valor: Currency);
end;
constructor TPedido.Create;
begin
 inherited;
 FBoleto := TBoleto.Create;
end;

procedure TPedido.FecharPedido(Valor: Currency);
begin
 FBoleto.Cobrar(Valor);
end;
```

Listagem 3. Emitindo o boleto de cobrança

```
var
 pedido: TPedido;
begin
 try
   pedido := TPedido.Create;
   pedido.FecharPedido(200);
   ReadLn;
 except
   on E: Exception do
    Writeln(E.ClassName, ': ', E.Message);
 end;
end.
```

Listagem 4. Interface IFormaPagamento

```
IFormaPagamento = interface
 ['{083BAF04-A68D-4148-8F7B-A10E44667CC3}']
 procedure Cobrar(Valor: Currency);
end;
```

Listagem 5. TBoleto implementando IFormaPagamento

```
TBoleto = class(TInterfacedObject, IFormaPagamento)
public
 procedure Cobrar(Valor: Currency);
end;

procedure TBoleto.Cobrar(Valor: Currency);
begin
 Writeln('Emitido boleto no valor de ' +
          FloatToStrF(Valor,ffCurrency,15,2));
end;
```

Figura 1. Resultado do exemplo de Injeção de Dependência

Listagem 6. TPedido usando o IFormaPagamento

```
TPedido = class
private
 FFormaPagamento: IFormaPagamento;
public
 constructor Create(formaPagamento: IFormaPagamento);reintroduce;
 procedure FecharPedido(Valor: Currency);
end;

constructor TPedido.Create(formaPagamento: IFormaPagamento);
begin
 inherited Create;
 FFormaPagamento := formaPagamento;
end;

procedure TPedido.FecharPedido(Valor: Currency);
begin
 FFormaPagamento.Cobrar(Valor);
end;
```

Listagem 7. Classe TCartaoCredito

```
TCartaoDeCredito = class(TInterfacedObject, IFormaPagamento)
public
 procedure Cobrar(Valor: Currency);
end;

procedure TCartaoDeCredito.Cobrar(Valor: Currency);
begin
 Writeln('Transacao no valor de: ' + FloatToStrF(Valor,ffCurrency,15,2) +
          ' autorizada pela operadora.');
end;
```

Listagem 8. Injetando a dependência

```
var
 pedido1: TPedido;
 pedido2: TPedido;
begin
try
   pedido1 := TPedido.Create(TBoleto.Create);
   pedido1.FecharPedido(200);
   pedido2 := TPedido.Create(TCartaoDeCredito.Create);
   pedido2.FecharPedido(150);
   ReadLn;
 except
   on E: Exception do
    Writeln(E.ClassName, ': ', E.Message);
end;
end.
```

E veja na **Listagem 8** como fica a criação dos Pedidos e a chamada do método FecharPedido.

O resultado desse código pode ser conferido na **Figura 1**. Note que as duas formas de pagamento são efetuadas, e podemos criar qualquer outra forma sem que haja a necessidade de modificarmos a classe TPedido.

Essa é uma das formas possíveis de se implementar a Injeção de Dependência. Note que estamos utilizando o construtor da classe para "injetar a dependência", podemos fazer isso de outras formas. Esse tipo de implementação nos permite inclusive, isolar as classes de Forma de Pagamento em uma DLL, para serem utilizadas em outras aplicações, se assim for o caso.

Dê seu feedback sobre esta edição!

A Clubedelphi tem que ser feita ao seu gosto. Para isso, precisamos saber o que você, leitor, acha da revista!

Dê seu voto sobre este artigo, através do link:

www.devmedia.com.br/clubedelphi/feedback

Delphi com estilo

Usando CSS no Prism

Neste artigo gostaria de sair um pouco do que sempre debatemos em outros artigos. Geralmente exibimos maneiras criativas de desenvolver softwares de qualidade. É claro que não poderíamos abrir mão de assuntos relacionados à Web. Como todos nós sabemos, a Web tem passado cada vez mais por transformações, principalmente hoje com o conceito de Cloud Computing, ter conhecimento de desenvolvimento voltado para a Web é um requisito obrigatório. E com a chegada do Delphi Prism ganhamos um forte aliado para esse fim, afinal utilizar o .Net Framework torna o nosso trabalho de desenvolvimento muito mais ágil. Ainda tratando-se de Web, temos visto aqui na Revista Clube Delphi uma série de artigos interessantes sobre desenvolvimento Web com Asp.Net e Delphi Prism, abordando a linguagem, definições dos conceitos de Web, ilustrando as diferenças entre o mundo Desktop e este novo mundo. É claro que tudo isto é importantíssimo, pois para desenvolver projetos de sucesso temos que focar na qualidade do software, e para isso não há como fugir de boas práticas de desenvolvimento.

Desenvolver Web sites é algo que envolve padrões, arte (visual), criatividade e qualidade (parte de codificação) e para unir tudo isso é óbvio que temos que conhecer sobre Web Standards. O foco do conceito de Web Standards é realizar uma padronização entre o desenvolvimento Web, permitindo que o conteúdo de seu Web site seja corretamente interpretado por qualquer tipo de Browser e para garantir que você possa desenvolver Web sites sem ferir estas normas. Porém, isto é teoria já que nem sempre os navegadores conseguem interpretar a codificação da mesma maneira e nem sempre os desenvolvedores seguem estes conceitos. Tratando-se ainda de Web Standards conheceremos neste artigo algo que hoje em dia é essencial para se desenvolver Web sites que unam os conceitos de padrões, arte e criatividade. Estamos falando do CSS.

CSS é a sigla de Cascading Style Sheets ou folhas de estilo em cascata em uma tradução livre. CSS é uma linguagem de estilo utilizada para descrever a forma de apresentação de um documento escrito ou de marcação como, HTML, XHTML ou XML. Hoje em dia o CSS está presente em quase todas as páginas que você visitar. Uma boa analogia para entendermos a aplicação do CSS em uma página é a seguinte: imagine que você deseje fazer uma escultura em argila. A princípio o que temos é apenas argila sem forma alguma, porém podemos utilizar ferramentas que nos ajudem a moldar esta argila, definindo formas e detalhes. Podemos utilizar tintas, pincéis e é claro, criatividade, e com isso poderemos criar nossa própria escultura. Assim como esta analogia, o CSS permite definir formas nos elementos de uma página, posicionar e dimensionar, criar efeitos, definir cores e o melhor de tudo

Resumo DevMan

De que trata o artigo:

O artigo trata de um dos conceitos mais importantes no desenvolvimento de layouts para Web, o CSS. Apresentaremos ao leitor o que é e qual a vantagem em se utilizar folhas de estilo em seu Web site. Faremos um exemplo com ASP.NET no Delphi Prism utilizando Master Pages e aplicando CSS.

Para que serve:

Serve para definições de estilo em páginas, permitindo a definição de padrões, formatação de objetos, centralização de código e apresentação de layouts. Através deste recurso, temos um típico exemplo de como utilizar o conceito de Web Standards, trazendo inúmeras vantagens ao desenvolvimento Web.

Em que situação o tema é útil:

Embora nem todos os Web sites sigam os Web Standards, é indispensável a utilização de CSS no atual cenário de desenvolvimento dentro dos padrões da Web. Qualquer Web site pode fazer uso de CSS e seus recursos, cabendo apenas aos Browsers o suporte destes recursos. Com o uso de CSS concentramos toda a parte de configuração de layout dos nossos Web sites em um arquivo externo ao HTML, facilitando assim a manutenção do projeto.

Delphi com estilo:

Neste artigo veremos um pouco sobre CSS, conheceremos seus diferentes tipos e como aplicar esta linguagem em uma página HTML, utilizando para isso o IDE do Delphi Prism / Visual Studio. A seguir, aplicaremos este mesmo estilo criado em um conceito de Master Pages, tornando o nosso trabalho muito mais ágil.

Nota do DevMan

O conceito de computação em nuvem (em inglês, cloud computing) refere-se à utilização de computadores e servidores compartilhados e interligados por meio da Internet, seguindo o princípio da computação em grade. O acesso a programas, serviços e arquivos é remoto, através da Internet - daí a alusão à nuvem.O PC torna-se apenas um chip ligado à Internet, a "grande nuvem" de computadores, sendo necessários somente os dispositivos de entrada (teclado, mouse) e saída (monitor).

isso, de uma maneira simples, organizada e centralizada.

Um dos requisitos para que você trabalhe com CSS é conhecer a linguagem HTML, afinal, estas regras são aplicadas diretamente aos elementos de seu Web site. O CSS como já falamos anteriormente, faz com que o desenvolvedor tenha todo o controle da página respeitando os Web Standards.

Quais as vantagens de se utilizar CSS?

Existem inúmeras vantagens em se utilizar e aplicar CSS em seus Web Sites. Dentre todas estas vantagens é importante ressaltarmos as que mais podem servir como incentivo para que você adote o CSS, são elas:

• Facilidade na manutenção do código.

Por se tratar de um "centralizador" de tags, a folha de estilo permite uma manutenibilidade muito eficaz, simples e prática, pois através dos seletores e classes é possível personalizar todo o seu documento de uma única vez. A grande sacada é que ao criar um novo objeto, uma div por exemplo, você pode fazer com que este objeto automaticamente receba as configurações definidas na folha de estilo apenas pelo simples fato de ser uma div, ou pode até mesmo utilizar atributos de identificação para especificar que esta determinada div será diferente das demais divs de seu site. Isto com certeza é um fator relevante para a utilização do CSS.

• Aumento da gama de funcionalidades visuais.

Como o HTML é uma linguagem simplificada, nem tudo que você fizer em CSS será compatível com o HTML. Porém, o CSS expande as funcionalidades do HTML permitindo uma personalização e inclusão de excelentes efeitos visuais.

• Criação de sites tableless.

Tabelas em HTML foram muito utilizadas para definir layouts na Web. A nova onda é desenvolver sites tableless. Mas o que seria isso? Trata-se de desenvolver Web sites sem, ou quase sem, utilizar tabelas (tag <table>). Utilizar divs no lugar de tables faz com que o seu Web site apareça melhor nos rankings dos sites de busca, pois isto faz parte dos mecanismos de busca, o tempo de download pode diminuir consideravelmente, sem contar na facilidade de manutenção de código. Esta s são só algumas das vantagens de se criar sites tableless.

• Diminuição do tempo de download.

Esta é também uma questão bem interessante. Como já falamos, os sites tableless podem tornar o tempo de download menor, porém a folha de estilos também auxilia na velocidade, já que o conteúdo da página será "cru", possuindo tags simples de HTML puro. Todas as definições de cores, estilo de fonte, posicionamento e tudo referente a layout, não estarão diretamente no código HTML, mas sim dentro do arquivo CSS. Com isso o carregamento do arquivo de estilos só será baixado uma única vez, fazendo com que nas próximas execuções o seu site carregue mais rapidamente.

• Permite o controle do modo de exibição.

Assim como dito anteriormente, o HTML é uma linguagem muito simples e como o CSS é um "expansor" do HTML, temos a possibilidade de definir o modo de exibição. Com isso conseguimos personalizar o Web site para diferentes tipos de mídia, fazendo com que ele seja corretamente interpretado em dispositivos móveis, em casos de impressão (onde deverá ser visualizado apenas o texto) e assim por diante. Através do CSS podemos ter este tipo de controle.

• Facilidade de expansão de funcionalidade.

Adaptar o seu site aos novos recursos de CSS, quando há alguma nova especificação pela W3C, é simples, já que a alteração será feita no arquivo de estilo e o conteúdo HTML não sofrerá nenhuma modificação. Vale lembrar aqui que o responsável pela visualização dos recursos de CSS é o Browser, e que caso o Browser não suporte aquele determinado recurso, ele simplesmente o desconsiderará, ignorando e não gerando nenhum erro, por este motivo é muito simples expandir as funcionalidades do CSS.

Especificações do CSS

Existe uma "divisão" denominada de especificações do CSS que possui três classificações atualmente. Com o conhecimento de cada uma das especificações, até suas pesquisas na Web por materiais de CSS podem se tornar mais fáceis, pois você buscará os recursos que deseja através destes tipos de classificação.

A primeira classificação, CSS1, é caracterizada basicamente pela definição de estilos, cores e tipos de fonte, espaçamentos e posicionamento de objetos. Posteriormente a W3C definiu novas especificações, denominando essas novidades como CSS2 que

a grosso modo expandem as funcionalidades do CSS1, principalmente em questão de posicionamento de elementos e tabelas. A grande novidade na linguagem é o padrão CSS3, que ainda tem sido definido pela W3C e logo será indispensável. Este tratará principalmente de efeitos, possibilitando criar formas, sombreamentos, gradientes, animações e muito mais. Na sessão links ao final deste artigo, você terá um link que ilustra alguns destes efeitos. Basicamente é uma ferramenta que possibilita criar estes efeitos e automaticamente criar o código correspondente do efeito para que você aplique conforme seu gosto.

A notícia ruim é que nem todos os browsers atuais dão suporte aos efeitos proporcionados pelo CSS3, e como a ideia é sempre desenvolver aplicações compatíveis com a maior quantidade de browsers possíveis é recomendável que se use com moderação.

Entendendo na prática

Se você já utiliza ou conhece a linguagem HTML, você não encontrará muitas dificuldades para aplicar CSS em suas páginas, no entanto, sempre vale relembrar algumas coisas. Vamos começar com exemplos de HTML puro, sem aplicação de estilos e no decorrer do artigo iremos começar a introduzir o CSS no HTML que fizemos, e então você compreenderá como é possível utilizar tanto HTML quanto CSS sem que para isso você necessite "poluir" seu código com várias tags de estilo.

O primeiro exemplo de HTML que faremos é algo comparado ao famoso "Hello World!", pois é algo muito simples. Primeiramente vamos criar uma página HTML pura, utilizando um bloco de notas por exemplo. Especificaremos o uso das tags <h>. Primeiramente crie um arquivo HTML com o código da **Listagem 1.**

Salve o arquivo como exemplo1.html. Ao executar a sua página no Browser você terá o resultado da **Figura 1.**

Note que o exemplo é algo extremamente simples. O que fizemos aqui foi demonstrar o uso das tags h1 à h6 dentro do HTML. Aqui vale ressaltarmos um detalhe interessante. Esta numeração

Figura 1. Executando o primeiro exemplo em HTML

das tags h1 à h6 é definida por grau de importância, logo teremos como mais importante a tag h1 utilizada como título da página e consequentemente como menos importante a tag h6. Atente para o fato de que toda tag que abre (<h1>) e fecha, (</h1>) deve possuir um conteúdo entre seu início e fim, diferentemente da tag <hr> por exemplo, que define uma linha horizontal na página e não requer conteúdo algum.

Vamos considerar este exemplo para trabalhar com as folhas de estilo. Antes de prosseguirmos com as alterações desta página, devemos compreender um pouco sobre os tipos de estilo.

Tipos de estilo

Existem três maneiras de se aplicar estilos em suas páginas. O primeiro tipo e mais poderoso é o estilo "linkado". Neste caso temos um arquivo independente da página com extensão .css na qual ficam todas as regras de posicionamento, links etc. A grande vantagem de se utilizar este modelo é que você pode ter múltiplos arquivos no seu Web site, ou mesmo reaproveitar o seu arquivo para outros Web sites que você venha criar.

Crie um arquivo chamado estilo.css no mesmo diretório em que foi criada a página anterior. Iremos personalizar aquele exemplo, fazendo algumas alterações. Após criar o arquivo estilo.css, adicione a seguinte instrução no mesmo:

```
h1
{
  font-family:Verdana;
  color:Green;
}
```

Listagem 1. Código HTML inicial

```
<html>
 <head>
  <title>Exemplo 1</title>
 </head>
 <body>
  <h1>Estou usando a tag h1</h1>
  <h2>Estou usando a tag h2</h2>
  <h3>Estou usando a tag h3</h3>
  <h4>Estou usando a tag h4</h4>
  <h5>Estou usando a tag h5</h5>
  <h6>Estou usando a tag h6</h6>
 </body>
</html>
```

Nota do DevMan

O World Wide Web Consortium (W3C) é um consórcio internacional com cerca de 300 membros, que agrega empresas, órgãos governamentais e organizações independentes, e que visa desenvolver padrões para a criação e a interpretação de conteúdos para a Web.

Como podemos ver no código, a estrutura de funcionamento da linguagem CSS é composta pelo elemento, neste caso h1 e dentro de chaves os atributos que desejamos setar para aquele determinado elemento. Neste caso alteramos a font para Verdana e a cor para verde. Faremos agora a chamada do estilo dentro do HTML. Neste primeiro caso faremos a chamada direta do arquivo que está no mesmo diretório, utilizando a tag <link> dentro do elemento <head>. Para o funcionamento é preciso configurar três atributos, conforme a seguir:

```
<link rel="Stylesheet" type="text/css" href="estilo.css"/>
```

Após termos referenciado o arquivo externo estilo.css, nosso HTML ficará como na **Listagem 2.**

Deste modo, salvando ambos arquivos e executando o exemplo no browser teremos o resultado mostrado na **Figura 2.**

Simultaneamente aprendemos dois conceitos ao aplicar este estilo. O primeiro conceito foi ilustrar o uso do CSS por meio de arquivos externos (links). O segundo conceito foi utilizar o seletor de elementos. Não entendeu aonde utilizamos o seletor de elementos? Vamos especificar mais neste mesmo exemplo sugerindo algumas alterações. Para entendermos melhor o conceito do seletor de elementos, vamos começar removendo o código das tags h2 à h6 e adicionando dois parágrafos. Veja como ficou o novo código HTML na **Listagem 3**.

Note que adicionamos dois parágrafos. Agora queremos alterar as cores da fonte de nossos parágrafos tornando os dois verdes como a tag h1. Para isso, vamos voltar ao arquivo de estilos e modificá-lo adicionando a seguinte instrução:

```
p
{
  color:Green;
}
```

Salve ambos os arquivos e abra o arquivo exemplo1.html em seu navegador. Você notará que os parágrafos ficaram com as fontes verdes como esperávamos, porém agora queremos apenas o primeiro parágrafo verde e o segundo na cor vermelha. Como proceder? Como você já deve ter notado, até aqui trabalhamos com os seletores de elementos diretamente nas tags. Este efeito é utilizado quando queremos padronizar um determinado elemento para toda a página, como no exemplo anterior que sugerimos que todos os parágrafos fossem verdes. Porém, quando existem particularidades podemos utilizar outras formas para aplicar a formatação.

A segunda maneira de se personalizar elementos é através da propriedade id. Todo elemento pode ou não possuir um id (identificador), e este deve ser único para cada elemento nomeado. O ideal é que quando for desenvolver um layout, você nomeie cada bloco com um id, como veremos mais adiante quando formos criar o layout da Master Page. Por enquanto vamos apenas nos preocupar no controle do elemento da página através do id. O primeiro passo é nomear o elemento com o id desejado, então altere o parágrafo 2 adicionando o id "paragrafo2" conforme o exemplo a seguir:

```
<p id="paragrafo2">Paragrafo 2: Seja bem vindo ao
nosso WebSite</p>
```

Com isso temos uma referência para aplicar estilos específicos a este elemento. Basta agora alterar o arquivo de estilo baseando-se neste id. Veja o exemplo que segue:

```
#paragrafo2 {
  color:Red;
}
```

Salvando ambos os arquivos e novamente abrindo o arquivo exemplo1.html você perceberá que conseguimos o efeito desejado, tendo o primeiro parágrafo na cor verde e o segundo na cor vermelha. Vamos dificultar um pouco mais para entendermos outra maneira de se personalizar elementos. Para isso criaremos novamente outro parágrafo abaixo do segundo e também vamos setá-lo com a cor vermelha. Modifique seu HTML deixando-o como na **Listagem 4.**

Note na **Listagem 4** que removemos o elemento id do parágrafo 2 e adicionamos um terceiro parágrafo ao código html. Se salvarmos neste momento o nosso html e executarmos no navegador, vamos perceber que os três estão na cor verde, devido ao nosso seletor da tag <p>, porém como desejamos o primeiro verde e os outros dois vermelhos vamos modificar o nosso arquivo .css. Poderíamos resolver este problema utilizando dois ids diferentes em cada um dos parágrafos, porém, esta não seria a forma correta, considerando que temos dois parágrafos diferentes com as mesmas características. A dificuldade

Nota

A aplicação de estilos em ids deve possuir o caractere "#" antes do seletor.

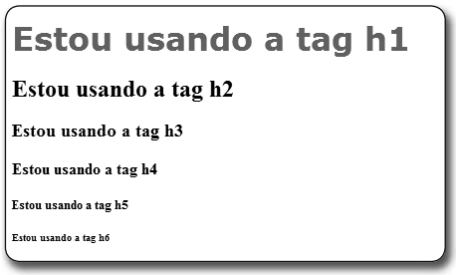


Figura 2. HTML modificado através de CSS externo

Listagem 2. Aplicando o CSS através de um arquivo externo

```
<html>
 <head>
  <link rel="Stylesheet" type="text/css" href=
   "estilo.css"/>
  <title>Exemplo 1</title>
 </head>
 <body>
  <h1>Estou usando a tag h1</h1>
  <h2>Estou usando a tag h2</h2>
  <h3>Estou usando a tag h3</h3>
  <h4>Estou usando a tag h4</h4>
  <h5>Estou usando a tag h5</h5>
  <h6>Estou usando a tag h6</h6>
 </body>
</html>
```

Listagem 3. Código HTML modificado

```
<html>
 <head>
  <link rel="Stylesheet" type="text/css" href=
   "estilo.css"/>
  <title>Exemplo 1</title>
 </head>
 <body>
  <h1>Estou usando a tag h1</h1>
  <p>Paragrafo 1: Compreendendo o conceito de
      seletores através da tag p </p>
  <p>Paragrafo 2: Seja bem vindo ao nosso
      WebSite</p>
 </body>
</html>
```

Listagem 4. Modificando o HTML

```
<html>
 <head>
  <link rel="Stylesheet" type="text/css"
   href="estilo.css"/>
  <title>Exemplo 1</title>
 </head>
 <body>
  <h1>Estou usando a tag h1</h1>
  <p>Paragrafo 1: Compreendendo o conceito de
      seletores através da tag p</p>
  <p>Paragrafo 2: Seja bem vindo ao nosso
      WebSite</p>
  <p>Paragrafo 3: Testando outra maneira de
      aplicar estilo a elementos</p>
 </body>
</html>
```

de manusear seria ainda muito maior se tivéssemos mais parágrafos para aplicar estilos, desta forma utilizamos a aplicação de estilos por meio de classes. Inicie modificando seu arquivo de estilo conforme a seguir:

```
.paragrafored
{
  color:Red;
}
```

Note que na instrução anterior definimos a classe de parágrafo através do caractere "." ponto. Deste modo, todos os elementos que desejamos que possuam estas especificações deverão possuir o nome da classe na propriedade class. Alteraremos a nossa estrutura HTML para ver o resultado no navegador. A **Listagem 5** contém o novo código HTML. Note como já dito o uso do atributo "class" dentro da tag <p> informando para a classe "paragrafored". O resultado você vê na **Figura 3**.

Com isso compreendemos o uso dos seletores de elementos. Agora continuaremos a explorar outras maneiras de se aplicar estilos em sua página HTML. Outra maneira de se aplicar estilos a uma página é utilizando os estilos embedded (incorporados). A vantagem é que, apesar de estar dentro da mesma página e não permitir a customização do Web site inteiro, ainda assim permite o controle dos elementos, porém apenas da página que está sendo aplicada. A desvantagem é que o seu arquivo HTML ficará maior, devido o fator das regras de estilo estarem embarcadas no próprio documento. Vejamos o exemplo da **Listagem 6**.

Um fator interessante que você pode notar na **Listagem 6** é que na linha 03 ainda temos a referência do arquivo "estilo.css", porém entre as linhas 04 e 08 inserimos a tag <style> que declara uma folha de estilo na própria página. Note na linha 05 que definimos a tag <h1> com a cor vermelha sendo que antes a mesma era verde. Tendo a cor verde definida primeiramente (na folha de estilo externa) e a cor vermelha definida de forma embarcada (na própria página), será considerada a cor vermelha como cor padrão para a tag <h1>. Veja na **Figura 4**.

Como já vimos até aqui, temos a opção de criar folhas de estilo externas (arquivos .css), embarcadas (dentro da própria página) e por último veremos a opção inline. Este é o método que traz mais desvantagens. Possibilita apenas a utilização no próprio HTML, deste modo limita um pouco os recursos do CSS e sobrecarrega muito a página. Geralmente é utilizada em conjunto com a tag para definir maneiras de personalizar o seu estilo (veremos um exemplo disso a seguir). Para exemplificar a aplicação deste último estilo, iremos modificar a tag <h1> para a cor amarela, deste modo deixe seu HTML como o da **Listagem 7.**

Observe na **Listagem 7** que ainda possuímos os dois estilos anteriores, tanto o estilo linkado/externo (linha 03), quanto o estilo embarcado (linhas 04 a 08). Perceba que a modificação foi realizada apenas na linha 12, onde definimos para a tag <h1> o atributo style="color:Yellow". É possível perceber que a sintaxe é praticamente a mesma para todos os estilos. Com isso você já está apto a ler, entender e escrever folhas de estilo de todos os modos, pois já entende os conceitos de funcionamento dos estilos. Um detalhe interessante a se comentar é que como você já deve ter notado,

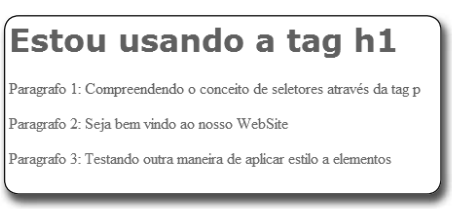


Figura 3. Resultado da aplicação de CSS usando a classe

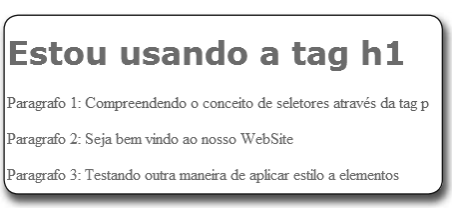


Figura 4. Predominância do estilo em cascata

Listagem 5. Novo código HTML

```
<html>
 <head>
  <link rel="Stylesheet" type="text/css"
    href="estilo.css"/>
  <title>Exemplo 1</title>
 </head>
 <body>
  <h1>Estou usando a tag h1</h1>
  <p>Paragrafo 1: Compreendendo o conceito de
   seletores através da tag p</p>
  <p class="paragrafored">Paragrafo 2: Seja bem
   vindo ao nosso WebSite</p>
  <p class="paragrafored">Paragrafo 3: Testando
   outra maneira de aplicar estilo a elementos</p>
 </body>
</html>
```

Listagem 6. Estilo de CSS embedded

```
01 <html>
02  <head>
03   <link rel="Stylesheet" type="text/css"
       href="estilo.css"/>
04    <style type="text/css">
05     h1 {
06       color:Red;
07     }
08    </style>
09   <title>Exemplo 1</title>
10  </head>
11  <body>
12   <h1>Estou usando a tag h1</h1>
13   <p>Paragrafo 1: Compreendendo o conceito
       de seletores através da tag p</p>
14   <p class="p">Paragrafo 2: Seja bem vindo ao
       nosso WebSite</p>
15   <p class="p">Paragrafo 3: Testando outra
       maneira de aplicar estilo a elementos</p>
16  </body>
17 </html>
```

Listagem 7. Estilo de CSS inline

```
01  <html>
02   <head>
03    <link rel="Stylesheet" type="text/css"
       href="estilo.css"/>
04    <style type="text/css">
05      h1 {
06        color:Red;
07      }
08    </style>
09    <title>Exemplo 1</title>
10   </head>
11   <body>
12    <h1 style="color:Yellow">Estou usando a
        tag h1  </h1>
13    <p>Paragrafo 1: Compreendendo o conceito
       de seletores através da tag p</p>
14    <p class="p">Paragrafo 2: Seja bem vindo ao
       nosso WebSite</p>
15    <p class="p">Paragrafo 3: Testando outra
       maneira de aplicar estilo a  elementos</p>
16   </body>
17  </html>
```

os estilos seguem um padrão "hierárquico" no qual predomina o estilo em linha. Se você possuir os três estilos em uma página, o estilo do elemento inline será o predominante. Portanto você já sabe que a ordem é: estilo linkado/externo>embarcado>inline. Com isso finalizamos a parte de apresentação dos tipos de estilo. Vamos trabalhar algo um pouco diferente agora. Iremos mostrar como definir a tag <h1> de título como uma imagem, pois geralmente sempre vemos nos sites a parte de título com o logotipo de nosso site. A questão é: "como fazer isso sem necessitar "sujar" as tags com atributos inline?". Vamos à resposta logo na criação da Master Page.

Criando a Master Page

Até aqui todos os exemplos que fizemos não requeriam necessariamente a utilização do Visual Studio (IDE do Delphi Prism), no entanto, se você achar mais prático pode criar o seu layout em outro editor qualquer, para só então passar o HTML pronto para o Visual Studio e consequentemente para a Master Page. Optaremos pelo Visual Studio neste artigo e já iremos criar o nosso modelo de Master Page. Acesse o menu File>New>Web site para iniciarmos nosso exemplo prático. Crie o seu site selecionando a linguagem Oxygene conforme o exemplo da **Figura 5.** Após ter selecionado o diretório apenas clique em OK.

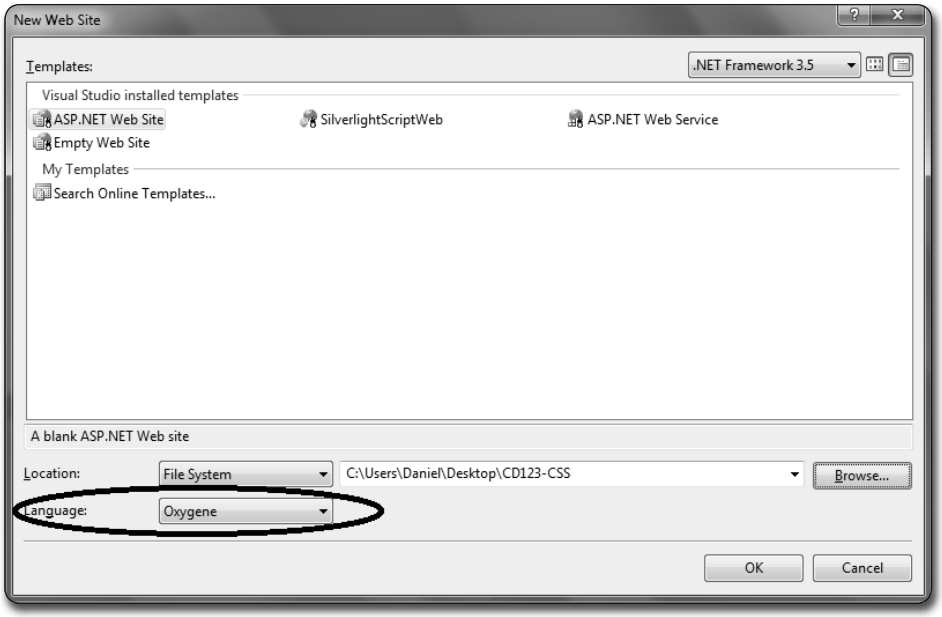
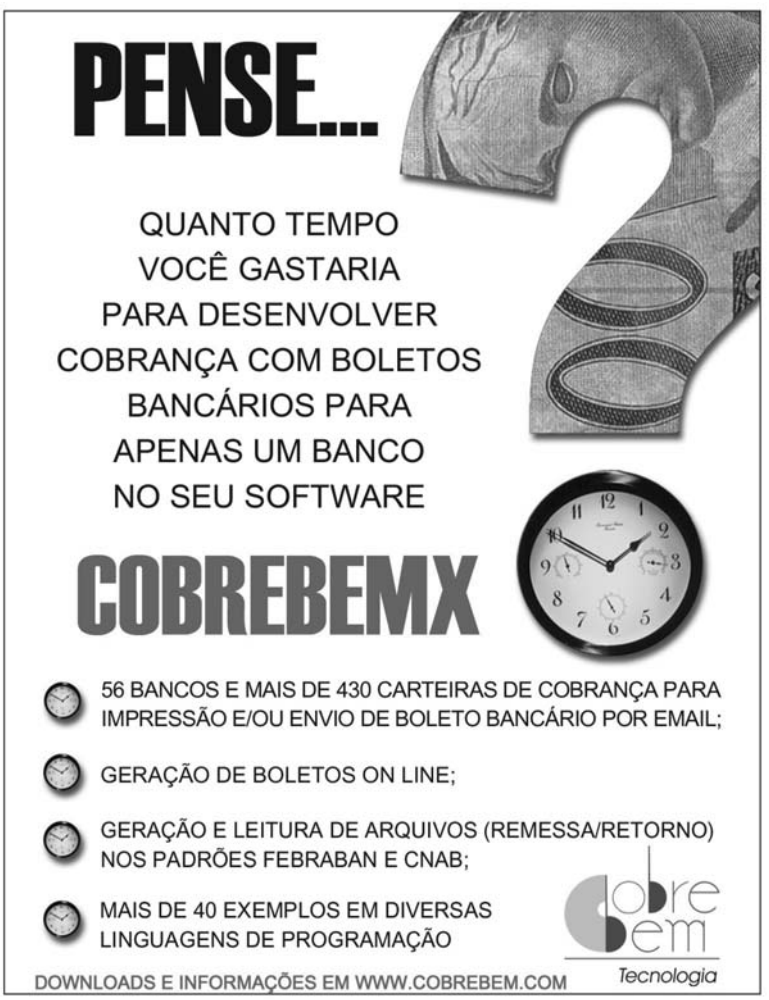


Figura 5. Criando a página principal

O primeiro passo a fazer é apagar o arquivo Default.aspx criado automaticamente pelo Visual Studio. Ele não será necessário para nós por enquanto, já que vamos criar diretamente a Master Page, portanto selecione o arquivo e com um clique de direita escolha a opção Delete. Em seguida clique novamente de direita sobre o Solution Explorer e clique na opção Add New Item em seguida certifique-se de ter selecionado as opções da **Figura 6,** clicando em seguida em Add**.**

Feito isso vamos dar início ao código de nossa página Master Page. Para nos poupar trabalho implemente o código aspx conforme a **Listagem 8.**

Vamos comentar o código da **Listagem 8**. O primeiro fator interessante neste



código encontra-se na linha 05 onde realizamos a chamada do nosso arquivo de folha de estilo, denominado "css.css", e que ainda não foi criado por nós. Note que estamos trabalhando com cinco elementos , que nada mais são do que containers para posicionamento dos elementos da página. Temos então uma chamada "geral" que será o container principal, dentro desta teremos as divs "cabecalho", "menu", "corpo" e "rodape". Perceba que dentro da div "cabecalho" temos os elementos <h1> e com um conteúdo " " (new backspace, ou novo espaço) que é um tanto quanto estranho, pois o título não possui conteúdo algum além de um espaço. Dentro do menu inserimos duas linhas através da tag <li>, estas encontram-se dentro do elemento <ul> que representa uma lista não ordenada (linhas 13 a 18). Note que o elemento ContentPlaceHolder, responsável pela exibição do conteúdo derivado da Master Page (páginas filhas), está dentro da div "corpo" (linhas 19 a 22), pois controlando esta div teremos o domínio do elemento ContentPlaceHolder automaticamente. O elemento "rodape" possui uma tag <p> que representa um parágrafo, conforme você pode ver na linha 24. Observe agora o resultado deste layout dentro do Visual Studio através da **Figura 7**.

Note que a **Figura 7** ilustra um layout totalmente pobre, sem estética alguma. Isto porque ainda não colocamos nossa folha de estilo para funcionar. Antes de prosseguir com o desenvolvimento da folha de estilo, abra o diretório onde foi criado o seu Web site e crie uma pasta com o nome de "img". Dentro desta pasta coloque uma figura qualquer, no meu caso vou inserir uma imagem do logotipo da ClubeDelphi dentro desta pasta. Esta imagem terá um tamanho aproximado de 400 x 240 px. Com isso você já terá a estrutura pronta para a adição da folha de estilos (que já foi incluída no Web Site). Crie fora do diretório img, mas no mesmo diretório do seu Web Site, o arquivo "css.css" e insira o código da **Listagem 9** dentro do mesmo.

O código da **Listagem 9** é o responsável pela "mágica" do layout. Embora seja um pouco longa, não é preciso nem numerá-la, pois a própria linguagem do CSS já nos propicia uma fácil compreensão. O grande fator necessário para se compreender este código é o entendimento dos seletores. Tudo que estiver iniciando em "#" é referente a um elemento que possui um id, ou seja, um identificador específico, conforme você já viu anteriormente. Logo na primeira definição onde temos a linha: body, html{} refere-se aos elemen-

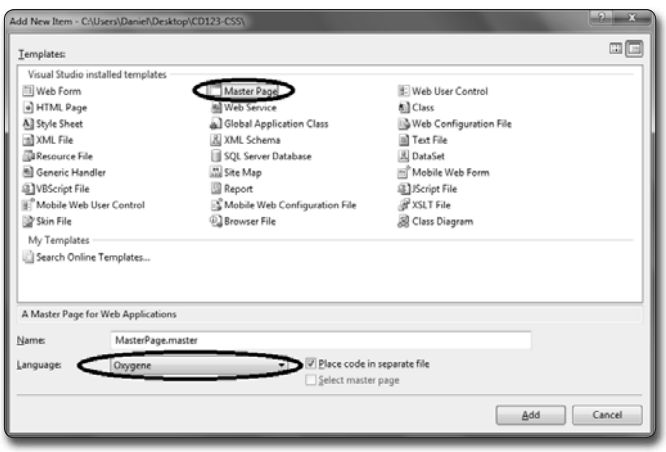


Figura 6. Configurando a Master Page

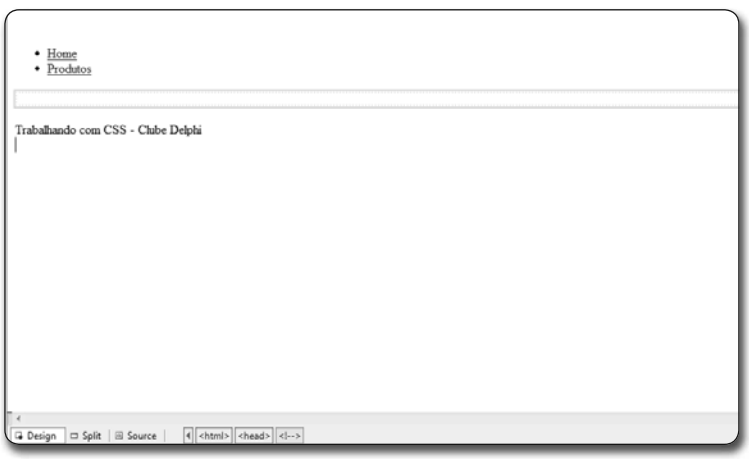


Figura 7. Layout da página no Visual Studio

Listagem 8. Código aspx da Master Page

```
01  <%@ Master Language="Oxygene" AutoEventWireup="true" CodeFile=
      "MasterPage.master.pas" Inherits="MasterPage" %>
02  <!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//
      EN""http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
03  <html xmlns="http://www.w3.org/1999/xhtml" >
04  <head runat="server">
05     <link rel="Stylesheet" type="text/css" href="css.css"/>
06     <title>Exemplo CSS - Clube Delphi</title>
07  </head>
08  <body>
09     <div id="geral">
10        <div id="cabecalho">
11           <h1><span> </span></h1>
12        </div>
13        <div id="menu">
14           <ul>
15              <li><a href="">Home</a></li>
16              <li><a href="">Produtos</a></li>
17           </ul>
18        </div>
19        <div id="corpo">
20           <asp:contentplaceholder id="ContentPlaceHolder1" runat="server">
21           </asp:contentplaceholder>
22        </div>
23        <div id="rodape">
24           <p>Trabalhando com CSS - Clube Delphi</p>
25        </div>
26     </div>
27  </body>
28  </html>
```

tos body e html, sendo assim, tudo que estiver separado por vírgulas, irá ser considerado como elementos que fazem uso das mesmas regras. Neste caso os elementos body e html que terão uma altura de 100% considerando o tamanho da página. Em seguida definimos as particularidades da div "cabecalho" onde definimos que a div terá a margem automática, largura de 760px, altura de 270px, borda pontilhada de 1px de largura e de cor cinza. Quando a margem está automática como neste caso, a largura de 760 é subtraída da largura da tela, então supondo que estamos desenvolvendo em um ambiente de 800x600, teremos: 800px-760px / 2 (lado esquerdo e direito), o que centralizaria a div em 20px para cada um dos lados. A segunda regra para entendimento desta listagem encontra-se a seguir, onde temos:

```
#cabecalho h1
```

Este código diz que estamos acessando o elemento h1 contido na div cabeçalho, ou seja, estamos tratando o seletor do elemento dentro do container. Neste caso foi configurada uma imagem de fundo, apontada para o diretório img, o modo de exibição display que diz que deve ser exibido como um bloco/container, altura de 270px e borda pontilhada com 1px de largura na cor cinza. Um detalhe interessante é visto como a seguir:

```
h1 span
```

Lembra-se que colocamos dentro da div "cabecalho" uma tag <h1> com um espaço em branco? O código anterior diz que toda vez que houver um elemento h1 seguido de um span, teremos a atribuição daquele estilo definido. Você poderia criar uma regra de classe, por exemplo, utilizando algo como .seletor e identificando na propriedade class do elemento, pois caso você tenha mais de um elemento span dentro de um elemento h1, você terá a aplicação da regra para todos eles. A seguir são definidas as regras referentes ao elemento "corpo" que como você pode observar é muito semelhante ao que já foi feito anteriormente no cabeçalho.

Note que foi definido um seletor #corpo p, que diz que qualquer parágrafo contido no corpo conterá font Tahoma. O mesmo ocorre com as definições do elemento #rodape, porém, com fonte Verdana e em cor branca. Os últimos códigos referem-se todos às tags contidas dentro do elemento #menu, tratando respectivamente o menu, ul, li e a tag <a>, que refere-se ao estado do link. Com isso você já pode deduzir todo o código que é praticamente autoexplicativo.

Finalmente, crie uma nova página derivada da Master Page, clicando de direita no Solution Explorer e escolhendo a opção Add New Item, escolhendo em seguida Web Form e selecionando a opção Select Master Page. A seguir, insira um texto qualquer dentro da tag <p> no ContentPlaceHolder, como mostra o código aspx da **Listagem 10.**

Nota

O código contido na tag <p> é apenas um texto próprio para popular conteúdo da Web. Na sessão links você confere o site oficial do gerador de texto.

Listagem 9. Código do arquivo .css

```
html, body
{
	height : 100%;
}

#cabecalho
{
	margin : auto;
	width: 760px;
	height : 270px;
	border : dotted 1px gray;
}

#cabecalho h1
{
	background-image : url(img/clubedelphi.jpg);
	display : block;
	width: 400px;
	height : 240px;
}

h1 span
{
	display : none;
}

#corpo
{
	margin : auto;
	width : 760px;
	border : dotted 1px grey;
}

#corpo p
{
	text-align : justify;
	font-family : Tahoma;
}

#rodape
{
	position : relative;
	width : 760px;
	margin : auto;
	background-color : Black;
	height : 30px;
}

#rodape p
{
	font-family : Arial;
	font-size : small;
	text-align : center;
	color : White;
}

#menu
{
	width : 760px;
	margin : auto;
	height : 28px;
	padding : 0px;
	border-style :none;
}

#menu ul {
	padding:0px;
	margin:0px;
	float: left;
	width: 100%;
	background-color:#EDEDED;
	list-style:none;
}

#menu ul li
{
	display: inline;
}

#menu ul li a {
	padding: 2px 10px;
	float:left;
	/* visual do link */
	background-color:#EDEDED;
	color: #333;
	text-decoration: none;
	border-bottom:3px solid #EDEDED;
}

#menu ul li a:hover {
	background-color:#D6D6D6;
	color: #6D6D6D;
	border-bottom:3px solid #EA0000;
}
```

Listagem 10. Código da página filha Default.aspx

```
<%@ Page Language="Oxygene" MasterPageFile="~/MasterPage.master"
  AutoEventWireup="true" CodeFile="Default.aspx.pas" Inherits="_Default"
  Title="Untitled Page" %>
<asp:Content ID="Content1" ContentPlaceHolderID="ContentPlaceHolder1"
  Runat="Server">
   <form id="form1" runat="server">
<h3>Lorem Ipsu</h3>
   <p>Lorem ipsum dolor sit amet, consectetur adipiscing elit. Ut tempor tempor
      metus eu porttitor. In sodales imperdiet eros, quis adipiscing eros congue nec.
      Pellentesque cursus accumsan tempus. Sed id felis et ligula placerat suscipit
      sit amet eget libero. Aliquam tincidunt elementum metus, vitae pellentesque
      ante facilisis eget. Pellentesque ullamcorper lorem nec mauris sollicitudin sed
      sagittis neque vestibulum. Nam eleifend lacus lectus, a eleifend lectus. Morbi
      a turpis purus, eu egestas sem. Duis ut nisi ante, vitae lobortis nibh. Aliquam
      erat volutpat. Lorem ipsum dolor sit amet, consectetur adipiscing elit.
      Pellentesque justo turpis, lobortis ac lobortis a, luctus id dolor. Sed suscipit mi
      ut urna feugiat nec malesuada mi dictum. Aliquam sed nisi et ipsum tincidunt
      congue. Integer eleifend elementum volutpat. Fusce pellentesque vestibulum
      risus, quis auctor justo accumsan eu. Donec vel ipsum id tortor fringilla pellen
      tesque a consequat tellus. Nullam porttitor gravida augue, in aliquet libero
      pharetra et. Suspendisse purus leo, sollicitudin vel tempor vel, vulputate condi
      mentum lectus. Nunc erat mi, sagittis ut egestas nec, imperdiet vel ipsum. Mae
      cenas ac enim sed nisl pretium mollis. Curabitur semper vehicula est, ut viverra
      purus ultricies a. Morbi id ligula sapien, sed mattis lectus. Donec et purus nisl, ac
      porta nisi. Sed molestie, dolor in posuere ultrices, nibh nunc pellentesque ante,
      mattis pretium eros ipsum sit amet eros.
   </p>
</form>
</asp:Content>
```

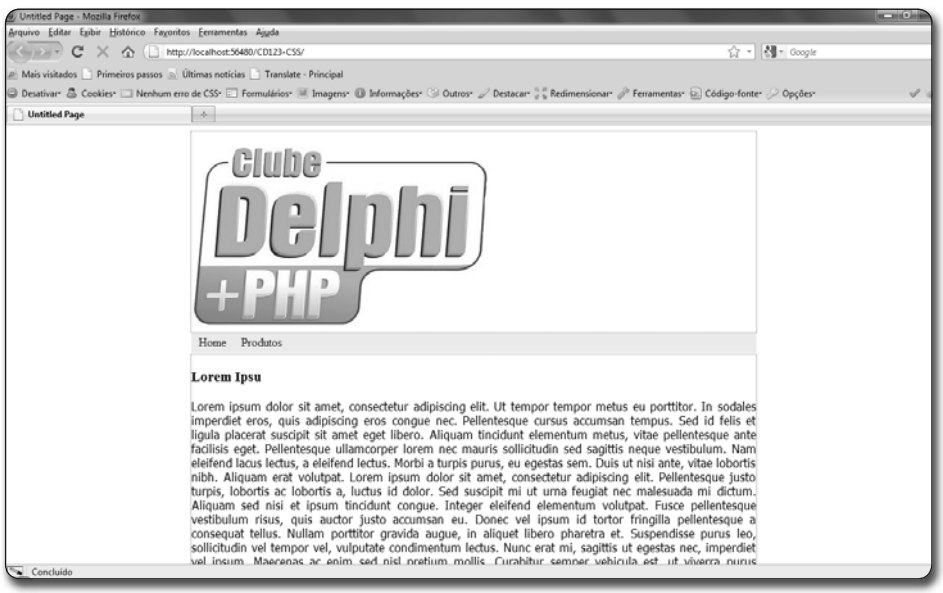


Figura 8. Layout final com CSS aplicado

Com isso temos tudo pronto. Execute o seu site e observe o resultado na **Figura 8.**

Conclusão

Como vimos, utilizar CSS pode nos trazer grandes benefícios. Aplicá-lo ao Delphi Prism através dos recursos de Master Page tornam o seu desenvolvimento ainda mais produtivo, de forma que será possível "replicar" o seu estilo definido para todas as demais páginas, definindo assim apenas o conteúdo que sofrerá alteração. Estudar CSS e Web Standards é uma necessidade básica para cada desenvolvedor que esteja interessado em ingressar no mundo da Web e criar/manusear seus próprios Web sites de uma forma inteligente, elegante, rápida e muito organizada. Espero que este artigo tenha despertado o interesse relativo a estes assuntos.

Daniel Sobrinho Laporte
daniel.sobrinho@devmedia.com.br
daniel.laporte@gmail.com
www.twitter.com/daniel_laporte
Colunista e editor técnico da Revista ClubeDelphi. Graduado em Sistemas de informação e pós-graduado em Administração de Bancos de Dados pela Universidade de Franca. É analista de sistemas e desenvolvedor Delphi desde 2005 desenvolvendo aplicações cliente/servidor e multicamadas.

Blogs do autor
http://twitter.com/daniel_laporte

CSS Maker – Utilizando CSS3
http://www.css3maker.com/

Zen Garden
http://www.csszengarden.com/

Gerador de texto Lorem Ipsum
http://www.lipsum.com/

Dê seu feedback sobre esta edição!

A Clubedelphi tem que ser feita ao seu gosto. Para isso, precisamos saber o que você, leitor, acha da revista!

Dê seu voto sobre este artigo, através do link:
www.devmedia.com.br/clubedelphi/feedback

FÓRUM DEVMEDIA

O lugar perfeito para você ficar por dentro de tudo o que acontece nas tecnologias do mercado atual

No novo Fórum da DevMedia você vai encontrar canais específicos de Delphi, ASP.NET, Java, Banco de Dados e Engenharia de Software; além de ter contato com profissionais qualificados da área para troca de informações, sugestões e muito mais.

ACESSE AGORA
www.devmedia.com.br/forum

DevMedia

Lazarus e FreePasc

Conheça o IDE Lazarus e o compilador FreePasc

Definitivamente o mundo Open Source não faz parte do cotidiano da maioria dos desenvolvedores Object Pascal. Ao contrário de desenvolvedores de outras linguagens que de uma forma ou de outra, já nasceram "abertas" como é o caso PHP e Java, por exemplo, os Delphianos não estão acostumados com as ferramentas e terminologias necessárias para acompanhar um projeto de código aberto. Também pudera, a criadora, antes chamada Borland, sempre cuidou muito bem de seus fieis súditos, e os Delphianos nunca precisaram utilizar algo além do próprio Delphi. O mecanismo de componentes reutilizáveis fez tanto o IDE quanto a linguagem Object Pascal se expandirem, e alcançarem uma legião de fãs, porém a Borland deu algumas "derrapadas", como por exemplo, com a criação do Kylix, onde muitos investiram tempo e energia no desenvolvimento multiplataforma e foram simplesmente abandonados com o final do projeto. As constantes mudanças de foco também assustaram a muitos, como por exemplo, com o Delphi 8 que previa o fim do Win32. Hoje em dia o Delphi está cada vez melhor nas mãos da Embarcadero que vem lançando versões frequentes e sempre cheias de novidades.

Na época em que o Delphi 8 assustou a todos, eu como amante de Object Pascal me senti frustrado e até mesmo traído com o andamento das coisas, e cheguei seriamente a pensar em aprender uma nova linguagem, como Java ou .NET. Mas nesse caso, perderia anos de conhecimento e código-fonte que disponho e, portanto logo descartei essa ideia. Passei então, a procurar uma solução Open Source que atendesse as minhas necessidades, pois desejava assim como os programadores de PHP, ajudar na construção e progresso de minha própria ferramenta. Foi então que decidi adotar o Lazarus/FPC. No início não foi nada simples, aliás, os primeiros testes foram decepcionantes. Eu sempre parava pra pensar em alguns fatores como: "Que lentidão para compilar!", "Por que o executável ficou enorme desse jeito?", "Cadê o ClientDataSet?". Mas com calma, muita pesquisa e deixando os preconceitos de lado, comecei a compreender melhor como as coisas funcionam no Lazarus, pude achar componentes equivalentes aos que eu usava. Após me animar com

Resumo DevMan

De que trata o artigo:

Dando continuidade ao artigo anterior, vamos agora, "turbinar" o Lazarus, instalando pacotes de componentes, todos muito úteis, e de código aberto. Aproveito para demonstrar recursos avançados do IDE, como o Refactoring de código. Vamos também conhecer um pouco mais sobre a organização das bibliotecas de componentes do Lazarus e como ele organiza o código-fonte do seu projeto. Para finalizar, forneço dicas úteis para quem é novo no mundo "OpenSource": Como usar o SVN e como e onde buscar ajuda necessária.

Para que serve:

Para adotar o Lazarus como ferramenta de trabalho, é fundamental conhecê-lo bem. Conhecer recursos avançados do IDE, conhecer como as bibliotecas de classes são organizadas, a organização dos fontes de um projeto etc. Também é fundamental compreender um pouco mais sobre como as coisas funcionam em projetos de código aberto. Isso lhe ajudará a decidir a adoção ou não dessa nova ferramenta que possui características ímpares ao Delphi, como por exemplo: ser OpenSource e Multiplataforma.

Em que situação o tema é útil:

Os primeiros passos no novo ambiente de desenvolvimento são sempre os mais difíceis. A decisão de migração de IDE e linguagem é algo muito importante, e que deve ser feito somente após muito estudo e cuidado. Para que a substituição do Delphi pelo Lazarus/FPC possa se tornar uma realidade, é fundamental conhecê-lo um pouco mais. Isso permitirá experimentá-lo mais a fundo, talvez iniciar a conversão de projetos simples, a fim de gerar os seus primeiros binários realmente úteis, com o Lazarus.

Lzarua e FreePascal:

Conheça mais a fundo o IDE do Lazarus e a organização de suas principais bibliotecas. Veja nesse artigo como instalar pacotes de componentes no Lazarus, para estendê-lo. Saiba como interagir com a comunidade de desenvolvedores do Lazarus e onde e como buscar ajuda.

Nota do DevMan

Para você que está iniciando agora em Delphi ou mesmo em Lazarus, talvez não tenha conhecido o projeto Kylix da antiga detentora do Delphi, a Borland. O projeto consistia basicamente em um IDE de desenvolvimento para o Sistema Operacional Linux, tudo isso utilizando o Object Pascal. Infelizmente o projeto foi descontinuado pela Borland, e hoje o IDE utilizado para desenvolvimento em Linux é o Lazarus. Uma das premissas das futuras versões do RAD Studio que a Embarcadero promete, tratam justamente do suporte para Linux e MacOS.

al

al, uma ótima alternativa Open Source – Parte 2

Daniel Simões Almeida

o sucesso da compilação dos primeiros projetos de testes e de ver meus primeiros programas rodando no Linux e WinCE, decidi me envolver um pouco mais e estudei melhor os fontes das bibliotecas do Lazarus, só então percebi o quão bem estruturados estavam.

Enfim encontrei no Lazarus/FPC uma chance de continuar com o meu sonho de produzir programas em multiplataforma, encontrei um projeto dinâmico e em pleno crescimento e muito mais do que isso, encontrei a chance de poder participar e colaborar com essa "boa ideia".

Nessa segunda parte do artigo, mostro como estender o IDE do Lazarus, afinal, um ambiente de desenvolvimento integrado não tem muita utilidade se não puder ser estendido e customizado. O Delphi se tornou um sucesso justamente por ter uma vasta gama de pacotes de componentes disponíveis, e com o Lazarus, isso não é diferente, existem vários pacotes de componentes, quase todos gratuitos, pois a maioria dos pacotes de componentes existentes foram migrados do Delphi e ajustados para o Lazarus/FPC. Vários deles inclusive compartilham os mesmos fontes, com a ajuda de pequenas diretivas de compilação (IFDEFs).

Nesse artigo, demonstro como instalar vários pacotes de poderosos componentes que facilitarão seu desenvolvimento e migração para o Lazarus. Vale salientar que alguns destes têm apenas o objetivo de modificar ou estender o IDE do Lazarus.

Também é de suma importância aprender mais sobre a etiqueta do mundo Open Source. Não podemos simplesmente escrever um e-mail para os "produtores" reclamando que o produto não funciona. Em Open Source isso não ajuda muito, mas podemos, ou melhor, devemos relatar os bugs encontrados e dar sugestões de melhorias. Mas como fazer isso de forma correta e apropriada? Também veremos como proceder com isto.

Espero que esta série de artigos faça você "balançar" e ver o Lazarus como uma opção real para ser a sua principal ferramenta de trabalho, assim como hoje é a minha. Mas por que eu desejo isso? Porque acredito que você tem muito a somar, e quanto maior o número de usuários, maiores são as chances do projeto receber ajuda e colaboração, de ter seus fontes estudados e consequentemente melhorados.

A instalação de pacotes no Lazarus é bastante diferente do Delphi. O Lazarus ainda não possui suporte a instalação de pacotes usando bibliotecas de ligação dinâmica, como ocorre no Delphi (.BPLs), ao invés disso, o pacote deve ser compilado de forma estática ao Lazarus, ou seja, todo o Lazarus deve ser recompilado a cada pacote instalado.

Isso pode parecer muito estranho para usuários Delphi, mas lembre-se que o Lazarus é distribuído com os fontes, inclusive os fontes do IDE, então a recompilação de todo o IDE não é um problema. Isso na verdade é uma vantagem, para quem acompanha as modificações do Lazarus pelo SVN, pois nesse caso, pode manter o IDE sempre atualizado com as últimas novidades. Entretanto, esse processo é um pouco lento, principalmente quando o IDE é compilado pela primeira vez. Mas isso não chega a ser um problema, pois a instalação de pacotes não é algo que fazemos todo dia.

Se você for instalar vários pacotes simultaneamente, não há necessidade de compilar o IDE a cada pacote. Você pode proceder com a instalação de todos os pacotes e compilar o IDE apenas no final.

Nota do DevMan

O Windows CE ou WinCE como é popularmente conhecido, trata-se de um Sistema Operacional próprio para a nova onda de dispositivos Tablets e também dispositivos embarcados. É comum vermos este Sistema Operacional em uso nos Netbooks devido a sua compatibilidade com as baixas características de Hardware. Para dispositivos como Smartphones é comum vê-los com outros tipos de plataformas, como Windows Mobile por exemplo.

Nota do DevMan

A versão do Delphi 8 teve como principal novidade o início do suporte ao .NET Framework. Embora houvesse muitas inovações como a mudança do visual do IDE, que continuou semelhante até a versão atual, o produto não teve grande sucesso como seu antecessor. As versões posteriores também possuíram suporte ao .NET, porém o sucesso do Delphi com .NET veio com força total com o lançamento do Delphi Prism, que possibilitou o suporte das versões mais atuais do .NET Framework utilizando o compilador Oxygene da RemObjects e uma "evolução" da linguagem Object Pascal.

Para isso, basta responder "Não" quando solicitado para proceder com a recompilação do IDE. O Lazarus irá marcar esse pacote para instalação, e procederá com a compilação de todos os pacotes marcados quando o IDE for recompilado.

Em alguns casos, o novo pacote instalado pode requerer alguma DLL que não está instalada no seu Sistema Operacional. Caso isso ocorra, o Lazarus não poderá ser executado. Você poderá ver uma mensagem com mais detalhes do erro, quando tentar executar diretamente o programa "Lazarus.exe" (que fica na pasta c:\lazarus). O atalho do Lazarus aponta para o programa "StartLazarus.exe", e por isso as mensagens de erro não são visualizadas quando ele é acionado. Para solucionar o problema, você deve providenciar as DLLs pendentes, ou desinstalar o pacote. Para desinstalar o pacote, execute o programa "Lazarus.old.exe", que é uma cópia do programa "Lazarus.exe" antes de ser recompilado.

Instalando o LazReport

O LazReport é um poderoso gerador de relatórios, criado a partir dos fontes do projeto FreeReport. Ele permite a geração de relatórios em tempo de execução, assim como ocorre no Rave Reports e Fast Reports.

Vários pacotes de componentes vêm pré-instalados no Lazarus. Outros são distribuídos com os fontes principais, porém, não são adicionados no IDE pelo programa de instalação. Você pode encontrá-los na pasta "C:\lazarus\components".

Com o IDE aberto clique em: "Pacotes>Abrir arquivo de Pacote (.lpk)". Navegue até a pasta: "C:\lazarus\components\lazreport\source" e abra o arquivo: "lazreport.lpk", conforme mostra a **Figura 1.**

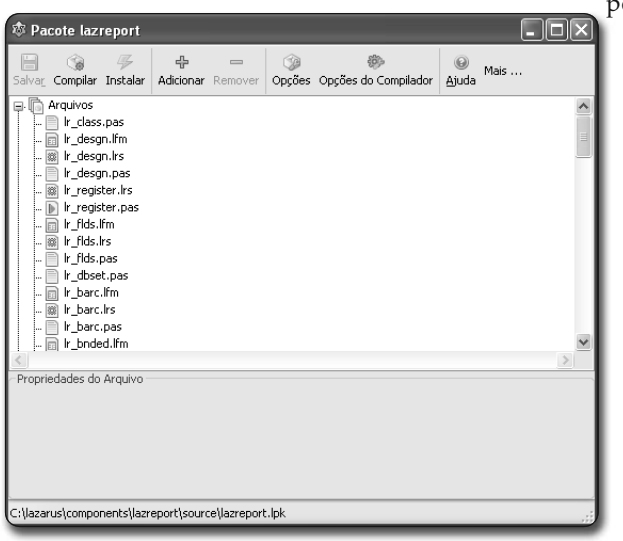


Figura 1. Abertura do pacote lazreport

Após a abertura do pacote, clique no botão "Compilar", e após o término da compilação, clique no botão "Instalar". O Lazarus irá exibir um aviso, informando que a compilação estática exige a recompilação de todo o IDE (**Figura 2**).

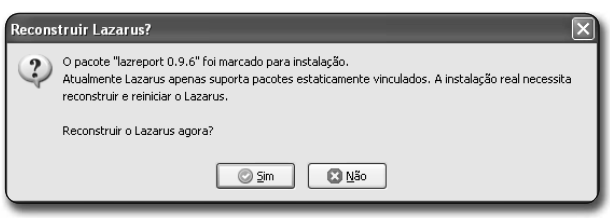


Figura 2. Mensagem de recompilação do Lazarus

Responda "SIM" e aguarde pacientemente a recompilação de todo o IDE (o que pode demorar alguns minutos). Quando o processo terminar, o Lazarus será fechado e aberto novamente. Quando o IDE reiniciar, você poderá ver na palheta de componentes, a aba do LazReport instalada, conforme mostra a **Figura 3**.

Figura 3. Paleta de componentes LazReport

Instalando pacotes que estendem o IDE

O Lazarus permite a criação de pacotes com a finalidade de estender o IDE, ou seja, não são pacotes de componentes reutilizáveis, mas sim programas, que se integram ao IDE, com a finalidade de torná-lo mais produtivo. O pacote "EditorToolBar" é um exemplo disso. Como o próprio nome diz, ele adiciona uma barra de tarefas na janela do editor do IDE.

Abra o pacote "editortoolbar.lpk" localizado em: "C:\lazarus\components\editortoolbar" e proceda da mesma maneira como descrito anteriormente, ou seja, clique em "Compilar", "Instalar" e aguarde a recompilação de todo o IDE. Observe que a recompilação do IDE foi bem mais rápida desta vez (isso ocorre porque os arquivos de compilação dos fontes do IDE (.ppu) já foram gerados no processo anterior). Após o termino da recompilação do IDE, observe na **Figura 4** que agora o seu editor de textos contém uma barra de tarefas, que pode ser configurada de acordo com a sua necessidade.

Onde eu configuro o LibPath dos componentes instalados?

Um problema comum para desenvolvedores Delphi é a falta de configuração da variável de ambiente "LibPath" após a instalação de um pacote de componentes. No Lazarus você simplesmente não precisa se preocupar com o LibPath. Cada pacote de componentes é responsável por informar o seu LibPath.

Quando o seu projeto fizer uso de algum Pacote de componente, todos os LibPaths desse pacote já serão herdados pelo seu projeto. O IDE automatiza todo esse processo. Quando você adiciona um componente a um Form do seu projeto, o

Nota do DevMan

Diretivas de compilação são fundamentais para a compatibilização de diferentes versões de IDE e são muito utilizadas em criação de componentes. Embora sejam mais vistas e utilizadas para esse meio, as diretivas de compilação permitem com que o compilador compreenda instruções que permitem desde a personalização do debug até a parametrização da aplicação.

IDE já adiciona a dependência do pacote ao qual esse componente pertence no seu projeto, e por consequência, o seu projeto "herdará" os LibPaths desse Pacote. Veja na **Figura 5** o que ocorreu após a inclusão de um componente TfrReport em um TForm vazio. Observe que a dependência ao Pacote "lazreport 0.9.6" (ou superior) foi criada no projeto.

Principais diferenças encontradas pelos desenvolvedores Delphi

É comum o desenvolvedor Delphi esbarrar em alguns problemas quando decide testar o Lazarus. Isso decorre do fato de que, apesar da incrível semelhança com o Delphi, o Lazarus é um produto totalmente diferente, que foi inteiramente escrito do zero, desde o compilador até o IDE. Portanto é natural esperar algumas diferenças na linguagem e IDE. Espero demonstrar aqui alguns dos pontos em que eu notei essa diferença. Caso a sua dúvida não seja respondida nesse artigo, você pode pesquisar a página Wiki do Lazarus, que tem seções inteiras dedicadas para conversão de programas escritos em Delphi para Lazarus (veja sessão Links ao final deste artigo).

Por que o executável fica tão grande ?

Isso ocorre porque o compilador FPC adiciona ao executável final várias informações responsáveis pela integração com o depurador utilizado atualmente pelo Lazarus, o GDB. O GDB é um famoso depurador de código aberto, distribuído pelo Projeto GNU. Um simples projeto com o form vazio pode atingir 12MB. Mas não se preocupe com isso, abaixo veremos duas maneiras de como você pode tornar o tamanho do executável "normal".

Usando o programa STRIP

O programa "strip.exe" é distribuído com o FPC, e tem a finalidade de remover informações desnecessárias do executável final. Ele será instalado pelo instalador do Lazarus na pasta dos binários do FPC, que geralmente é: "C:\lazarus\fpc\2.4.3\bin\i386-win32". Assim como o compilador FPC, ele é um programa de "linha de comando", e deve ser executado em um script (.BAT) ou em uma janela MS-DOS. Apesar de inúmeras opções, seu uso é bastante simples. Digite:

```
strip.exe SeuExecutavel.exe
```

Note que um projeto de "Form vazio" caiu de 12MB para apenas 1,5MB.

Configurando o projeto para não incluir informações de Debug

Provavelmente você achará essa opção bem mais simples. Mas lembre-se que uma vez selecionada, o Debugger não mais funcionará com o seu projeto até você remover essa opção e recompilar o seu projeto novamente.

Entre em Projeto>Opções de Projeto, na nova janela que se abrir selecione a opção "Vinculação" (ou Linking) e desmarque a opção: "Exibir número de linhas..." e marque a opção "Remover símbolos do executável...", como mostra a **Figura 6**.

Após isso compile normalmente o seu projeto, e você notará que o tamanho do seu executável ficará bem menor. Essa configuração é útil para gerar o executável final, que será distribuído para o usuário. Mas lembre-se de desfazer essas configurações quando necessitar depurar o seu projeto.

Por que a compilação é mais lenta?

A compilação é bem mais lenta que no Delphi. O compilador do Delphi é uma "obra de arte" e será difícil o FPC se equiparar ao Delphi no quesito velocidade de compilação. Porém, lembre-se que o FPC é um compilador multiplataforma, ou seja, ele não é otimizado apenas para Windows. Porém, o código gerado pelo FPC não perde nada em desempenho.

Os desenvolvedores do Lazarus estão atentos a isso, e a velocidade de compilação melhorou sensivelmente desde a versão 0.9.24. E é claro, quanto mais memória e CPU você tiver, mais rápidos o processo de compilação e linkagem serão.

Nota do DevMan

O Subversion ou SVN como é conhecido, é um dos mais populares sistemas de controle de versão. Hoje é uma ferramenta que integra-se aos mais famosos IDEs disponíveis no mercado, como Visual Studio e NetBeans. O crescimento da ferramenta tem sido tão popular que recentemente ganhou espaço entre as novidades da nova versão do Delphi, denominada de RAD Studio XE. Com ele é possível realizar todo o controle de versionamento de código-fonte evitando conflitos de units e realizando merge de alterações neste caso. Vale lembrar que a ferramenta é Open Source.

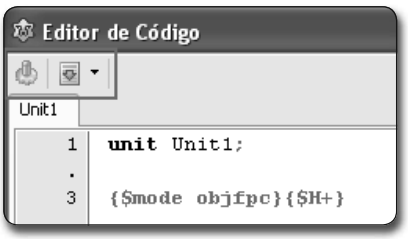


Figura 4. Barra de tarefas integrado ao IDE

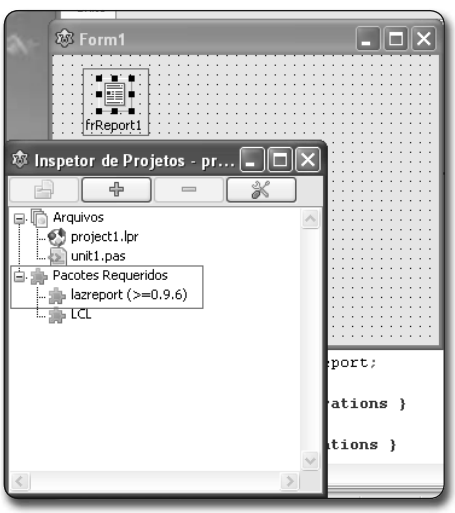


Figura 5. Informando o LibPath automaticamente no projeto

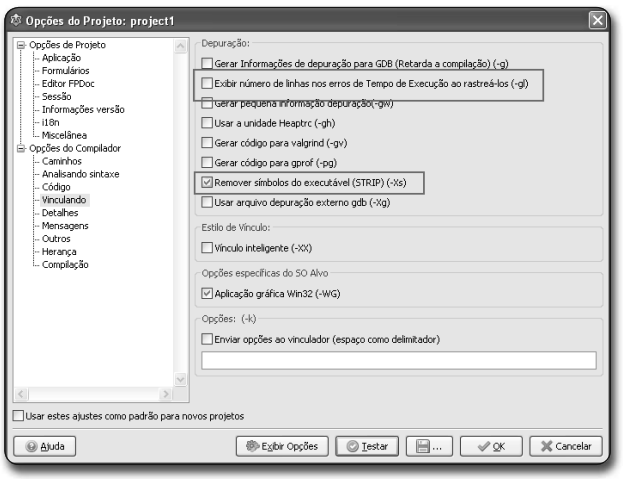


Figura 6. Configurações auxiliares

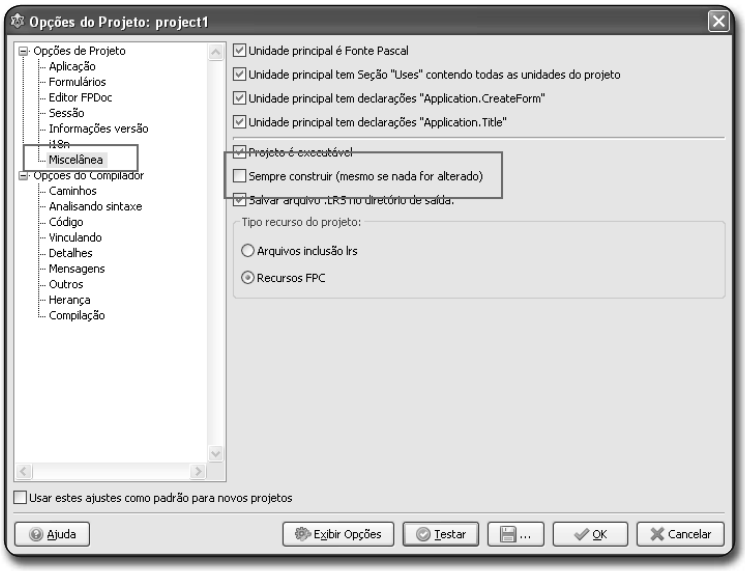


Figura 7. Configurando a opção de Link otimizado

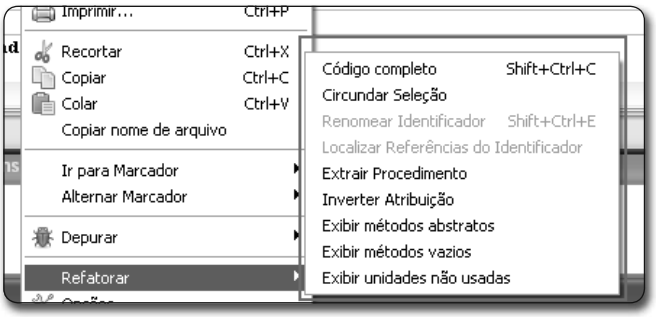


Figura 8. Opção de refatoração de código

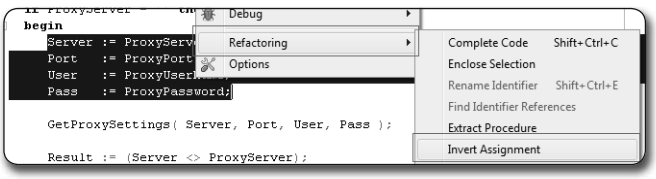


Figura 9. Opção de inverter atribuição

Você também pode configurar o IDE para evitar que ele rode o Link cada vez que você executar o seu projeto pelo IDE, isso no caso de você não ter modificado nada nos fontes. Acesse o menu Projeto>Opções de Projeto, clique na opção "Miscelânea" e desmarque a opção "Sempre construir..." (**Figura 7**).

Refactoring de código

A refatoração de código é um "show à parte". O menu de refatoração pode ser acessado através do botão direito do mouse no editor de texto (**Figura 8**).

Com a refatoração você pode organizar melhor o seu código-fonte. O menu de refatoração possui diversas ferramentas para finalidades distintas, mas que ajudam muito na realização de algumas tarefas "enfadonhas".

Inverter Atribuição

Suponha que você possui uma longa lista de atribuições a variáveis de trabalho, e após executar uma determinada rotina, deseja restaurar os valores das variáveis de trabalho às variáveis originais, ou seja, efetuar a operação inversa de atribuição. Apenas copie o código já escrito, selecione-o e use a opção "Inverter Atribuição". Veja as **Listagens 1** e **2** e a **Figura 9.**

Extrair Procedimento

Outro recurso que utilizo muito é a extração de procedimentos. Ele permite criar novos procedimentos ou funções a partir de um trecho de código-fonte previamente selecionado. Isso é muito útil quando você percebe que precisará de determinado trecho de código em outra rotina, e para não repeti-lo, decide criar uma nova função ou procedimento para que ele possa ser chamado de vários lugares.

Exemplo: Suponha que desejamos extrair o trecho de código-fonte selecionado na **Figura 10**, e criar um novo procedimento chamado "AtribuirNome". Após selecionar o código-fonte e entrar nas opções de menu "Refatorar>Extrair Procedimento", visualizaremos a tela também da **Figura 10.**

Após clicarmos no botão "Extrair" o IDE irá criar um novo procedimento privado (conforme a opção que selecionamos), e substituirá o código-fonte do bloco, por uma chamada a esse novo procedimento, que agora poderá ser chamado de outros lugares, veja o resultado na **Figura 11**.

Listagem 1. Código original

```
Server := ProxyServer
Port   := ProxyPort;
User   := ProxyUserName;
Pass   := ProxyPassword;
```

Listagem 2. Código invertido

```
ProxyServer   := Server;
ProxyPort     := Port;
ProxyUserName := User;
ProxyPassword := Pass;
```

O que é e como usar o SVN?

Devido às características do Lazarus, praticamente todos os componentes disponíveis para ele são distribuídos também em regime de código aberto, e muitas vezes através do SVN. SVN é um sistema de controle de versão usado por milhares de desenvolvedores. Os sistemas de controle de versão visam simplificar a colaboração entre equipes e ainda controlar o histórico de atualizações dos arquivos. O SVN guarda todas as alterações em cada arquivo como uma versão numerada e sempre que necessário, podemos comparar ou até mesmo voltar para uma versão anterior, sem deixar de ter a versão mais atual guardada do repositório (servidor do SVN).

Muitos projetos de componentes disponíveis para o Lazarus não são distribuídos em arquivos compactados (ZIP), mas sim através do SVN, ou seja, os desenvolvedores dos componentes permitem livre acesso de leitura ao repositório de fontes do SVN. Porém, apenas os desenvolvedores do componente possuem direito de enviar alterações no SVN (opção de commit).

Para baixar os fontes de um repositório SVN você precisará de um cliente SVN. O mais famoso e simples de usar, é sem dúvida o TortoiseSVN, que também é de código aberto. Ele se integra com o Windows criando novas entradas no menu

pop-up do Explorer, e sinalizando o estado dos arquivos com mini ícones.

Se você escolheu a língua portuguesa na interface do TortoiseSVN, quando falamos em "commit", entenda "gravar"; quando falamos em "update", entenda "atualizar" e quando falamos em "checkout" entenda "obter módulo".

Para baixar os fontes de um projeto usando o TortoiseSVN siga os seguintes passos:

• Crie um novo diretório para receber os fontes. Mas procure evitar a criação de subpastas dentro do diretório do Lazarus, pois isso dificultará a instalação de uma nova versão do mesmo;

• Usando o Windows Explorer, clique com o botão direito sobre o diretório criado e escolha a opção: "SVN Checkout";

• Entre com a URL de download do repositório;

• Clique em OK, e aguarde o término do download.

Nas próximas vezes que desejar atualizar os fontes, não será necessário efetuar os passos descritos anteriormente, basta clicar com o botão direito no diretório e usar a opção: "SVN Update". Nesse caso, somente os fontes modificados ou novos serão baixados.

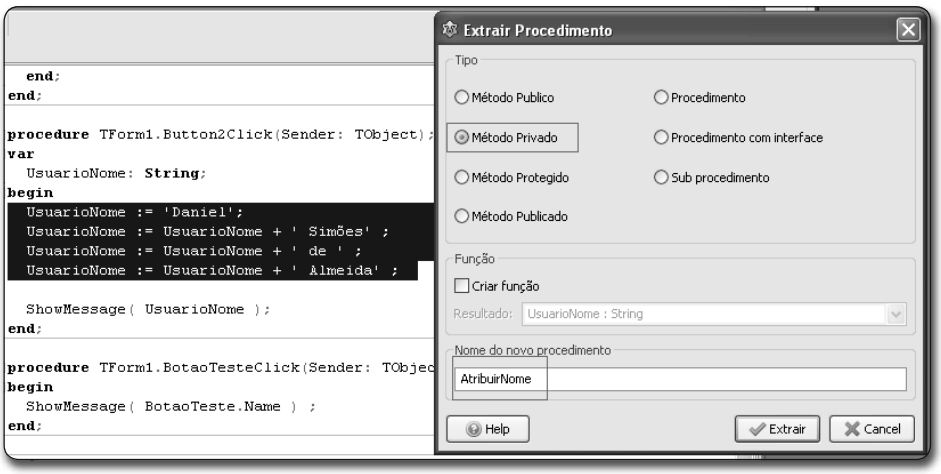


Figura 10. Seleção de código e extração de procedimento privado

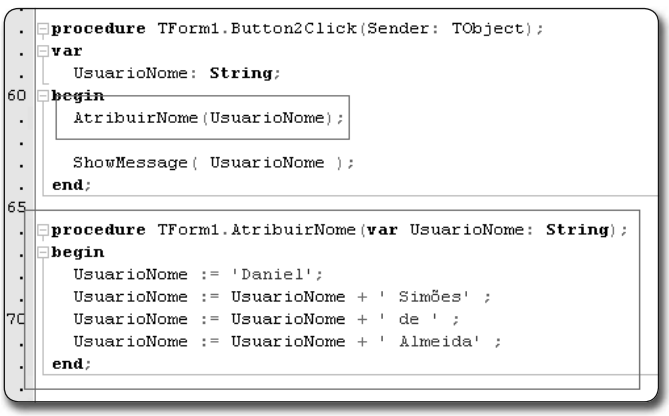


Figura 11. Método criado através do procedimento de extração

Baixando e instalando os componentes da CCR

O CCR é um subprojeto do Lazarus, destinado a ser um repositório centralizado de componentes e códigos (Code and Component Repository). A página principal do projeto você encontra na sessão Links ao final deste artigo. Vários desenvolvedores atuam no CCR, cada desenvolvedor é autônomo (e voluntário) e portanto cuida apenas de sua "pasta", ou seja, do seu componente.

Na CCR você encontrará componentes importantes como, por exemplo: RX, Web Service ToolKit, PowerPDF, JVCLLaz, ZVDateTimeControls, VirtualTreeView, e diversos outros componentes para as mais diversas finalidades. Muitos dos componentes existentes no CCR são traduções de componentes famosos existentes para o Delphi, como por exemplo, o VirtualTreeView que é mantido pelo brasileiro Luiz Américo.

Todos os componentes do CCR são de código aberto, porém suas licenças de uso são variadas, conforme o desejo de seu autor original. Acredito que todos os componentes do CCR são multiplataforma, ou seja, podem rodar também em Linux e MacOS. Mas convêm antes consultar a página do componente para conferir a compatibilidade.

Devido o fato de cada componente ser mantido por um desenvolvedor autônomo, a forma como ele é distribuído e atualizado pode variar. Alguns desenvolvedores disponibilizam pacotes compactados no site SourceForge, enquanto outros, atualizam apenas o repositório do SVN. A maioria dos componentes do CCR possui uma página própria com informações detalhadas sobre o mesmo. Esta reside dentro das páginas Wiki do Lazarus, e você poderá acessá-la através do endereço disponível também na sessão Links. Todos os componentes da CCR possuirão um ou vários arquivos de pacote para serem instalados no Lazarus.

Baixando e instalando a ZeosLib

A ZeosLib é um poderoso conjunto de componentes para acesso a banco de dados. Ela é compatível com Delphi e Lazarus, e também é multiplataforma. Suporta vários bancos de dados como MySQL, MySQL embedded, FireBird, FireBird embedded, Interbase, PostgreSQL, Oracle, SQLite e Sybase.

O Lazarus já possui um completo conjunto de componentes de acesso a banco de dados, que permitem acessar nativamente alguns bancos de dados, como por exemplo: PostgreSQL, Oracle, MySQL, FireBird, SQLite, DBF, além de ODBC. Esses componentes acompanham o Lazarus, e normalmente já são instalados por padrão. Caso eles não constem no seu IDE, instale o Pacote "sqldblaz.lpk" contido no diretório "C:\lazarus\components\sqldb". Para suporte nativo a DBFs, procure pelo pacote "dbflaz.lpk".

Se o Lazarus já possui componentes de acesso a banco de dados, por que devemos instalar a ZeosLib? Não é necessário e

Figura 12. ZeosLib instalado no Lazarus

Nota

O link para o download de todos os componentes encontra-se disponível na sessão Links ao final deste artigo.

sim opcional. A sugestão que dou para a instalação é apenas baseada em minha opinião, que considero a ZeosLib superior aos componentes nativos do Lazarus, além de ser mais coesa e fácil de usar em relação aos componentes nativos do Lazarus.

Se você já usava a ZeosLib no Delphi, se sentirá "em casa", mas se você usa o DBExpress, terá um pouco de dificuldade inicialmente, devido às diferenças de uso. Em relação à maneira de programação e uso, a ZeosLib é muito semelhante a outros componentes de acesso a banco de dados já existentes no Delphi, como por exemplo os antigos componentes do BDE. Após fazer o download do arquivo, descompacte-o em uma pasta de sua preferência. Abra todos os pacotes contidos no diretório "\Zeos\packages\lazarus", você pode selecionar vários pacotes simultaneamente na janela de diálogo de abertura.

Com exceção do pacote "zcomponent.lpk", todos os demais são apenas pacotes de runtime, portanto não necessitam de ser instalados, mas, precisam ser compilados. Portanto, clique em "Compilar" e em "Todos os Pacotes", deixando por último o pacote ZComponent. Observe na janela de mensagens se a compilação dos pacotes ocorreu sem problemas.

Quando chegar no pacote "ZComponent", além de clicar em "Compilar", clique em "Instalar" e observe que o Lazarus resolve as dependências e já adiciona na lista de pacotes a serem instalados os pacotes de runtime compilados anteriormente. Reconstrua o Lazarus quando solicitado, e pronto, a ZeosLib será instalada. Veja a paleta Zeos Access conforme a **Figura 12**.

Onde encontro mais componentes?

Projeto ACBr

Como idealizador e principal mantenedor do Projeto ACBr eu não poderia deixar de mencioná-lo. O projeto ACBr é de código aberto, compatível com Delphi e Lazarus, e multiplataforma. O objetivo do projeto é ser uma coletânea de componentes específicos para aplicações de automação comercial. Com o ACBr você conseguirá acessar diversos equipamentos de automação comercial de forma direta (sem a necessidade da DLL do fabricante), como por exemplo: ECF (emissores de cupom fiscal, balanças, impressoras de cheque, gavetas, e outros). Outros destaques do projeto são os componentes para nota fiscal eletrônica (NFe) e transferência eletrônica de fundos (TEF).

A página do projeto você encontra na sessão Links. Na própria página você encontrará instruções de como efetuar o download dos fontes pelo SVN, e também os endereços do fórum do ACBr para suporte.

Fortes Report

O Fortes Report é velho conhecido dos usuários Delphi. Criado por brasileiros, ele é um excelente gerador de relatórios. Foi portado para o Lazarus graças ao esforço de vários usuários da comunidade Lazarus no Brasil, como Isaac Trindade e Luiz Américo. Você pode fazer download por SVN através da URL disponível ao final do artigo.

ExtPascal (Web)

O ExtPascal também é um projeto compatível com Delphi e Lazarus. Ele permite criar aplicações Web, fazendo uso intensivo de AJAX, através do Framework ExtJS.

Synapse (TCP/IP, Serial)

A Synapse é um conjunto de classes para comunicação em baixo nível de TCP/IP e Serial. É compatível com Delphi e Lazarus e também é multiplataforma. Se você possui código-fonte em Delphi dependente da Indy, considere seriamente adaptá-lo, para substituir a Indy pela Synapse. Isso porque instalar a Indy no Lazarus apesar de possível não é tarefa simples, e provavelmente você esbarrará em alguns bugs de difícil resolução, pois não há um suporte "oficial" da Indy para o Lazarus. A Synapse é capaz de realizar praticamente tudo o que é feito pela Indy, e é extremamente simples de usar, além de ser bem mais "leve". Nós do projeto ACBr, dependemos muito da Synapse (que é distribuída com os nossos fontes). Está é a principal responsável por toda parte de comunicação TCP/IP, envio de e-mail, conexão segura, e comunicação serial do projeto ACBr.

Conhecendo as extensões dos arquivos gerenciados pelo Lazarus

Apesar de compartilhar com o Delphi a extensão de arquivo .pas, o Lazarus possui outros sufixos de arquivos, desconhecidos dos usuários Delphi. Na **Tabela 1** são relacionados os tipos mais comuns e para que servem.

WidgetSets

Um WidgetSet é um conjunto de controles ou componentes visuais, que compõem uma interface gráfica com o usuário (GUI). Todas as aplicações visuais são compostas de vários Widgets, como por exemplo: o Edit, ComboBox, RadioButton, Grids etc. Para que o Lazarus possa ser multiplataforma, ele deve ser capaz de suportar outros conjuntos de Widgets, além do disponível no Windows (Win32). Portanto, o código-fonte dos componentes do Lazarus não são ligados diretamente a uma API de componentes (WidgetSet) específica. Ao invés disso, você pode compilar o Lazarus e suas bibliotecas para um Widget de sua preferência.

Atualmente o Lazarus suporta os seguintes Widgets: Win32, WinCE, GTK, GTK2, QT, Carbon, fpGUI, além de outros mais exóticos. Nem todos eles estão completos e alguns exigem um Sistema Operacional específico. Ao final do artigo você terá um Link que pode lhe auxiliar no aprendizado sobre cada Widget.

O Lazarus assume um Widget padrão, dependendo do Sistema Operacional em que ele esteja sendo executado. No Windows, obviamente, ele usa o Win32 e a aplicação fica com uma ótima aparência, bem integrada ao tema do Sistema Operacional. No Linux, o Lazarus utilizará a GTK2 como Widget padrão e no MacOS o Carbon.

Também é possível usar um Widget diferente do padrão, apesar deste fator requerer um conhecimento mais avançado do Lazarus. Por exemplo, no Windows podemos usar GTK2, ou QT, mas para isso, precisamos instalar no Windows as DLLs dessas bibliotecas, além de recompilar todas as bibliotecas do Lazarus para o Widget escolhido. Em alguns casos, é necessário usar compilação cruzada (cross-compiling), como por exemplo, no caso da Interface com WinCE, onde usamos o Lazarus em Windows para gerar código nativo para WinCE.

As bibliotecas do Lazarus

O Lazarus basicamente é dividido em três grandes bibliotecas, a seguir faço um breve resumo sobre elas. O IDE já resolverá as dependências do seu projeto, em relação a essas bibliotecas.

A RTL

A RTL é a Runtime Library, ou biblioteca de tempo de execução. Ela está mais associada ao compilador (FPC) do que com o IDE (Lazarus). Ou seja, atualizar os fontes do Lazarus pelo SVN não irá atualizar a RTL, para isso, você precisaria atualizar os fontes do FPC pelo SVN (algo complexo de ser feito, pois o FPC precisaria ser recompilado). Se você deseja atualizar o FPC ou a RTL, é mais simples baixar uma nova versão do SnapShot e reinstalar o Lazarus. Os fontes da RTL geralmente são instalados em: "C:\lazarus\fpc\2.4.3\source\rtl".

Toda aplicação compilada com o Lazarus será dependente da RTL. Ela tem a finalidade de prover serviços básicos, como por exemplo:

- Inicializar a própria aplicação;
- Prover funções padrões do Pascal como, por exemplo, Write, WriteLn;
- Rotinas matemáticas (não resolvidas pelo compilador);

A FCL

A "Free Component Library" foi criada para ser um equivalente de código aberto da VCL do Delphi, porém, todos os componentes desta biblioteca são não visuais. Ela é mantida pela equipe do FPC de maneira que para atualizá-la é necessário atualizar o FPC, e não somente o Lazarus. A ideia da FCL é prover um conjunto completo de classes, procurando manter a máxima compatibilidade do código-fonte escrito em Delphi, permitindo o reaproveitamento de código-fonte Delphi no Lazarus.

A LCL

A "Lazarus Componente Library" implementa um conjunto de componentes visuais (Widgets), destinados a criação de aplicações gráficas (GUI). Ela é mantida pela equipe do Lazarus. A atualização do Lazarus pelo SVN atualizará a LCL, que após isso deverá ser recompilada. Esta possui seus fontes divididos em duas partes, a LCL e as Interfaces. Isso permite ao Lazarus suportar vários Widgets. Os fontes que são compartilhados e independentes de Widgets encontram-se em: "C:\lazarus\lcl", e toda programação específica para um Widget fica na pasta: "C:\lazarus\lcl\interfaces". A LCL deve ser previamente compilada contra um WidgetSet para que possamos utilizá-lo.

Onde obtenho ajuda ou suporte?

Hoje em dia, você não terá dificuldades para encontrar ajuda ou artigos para Lazarus. Há até mesmo opção de suporte pago. A comunidade de usuários de Lazarus, apesar de bem menos numerosa que a de Delphi, é bastante solícita. O fato do Lazarus ser um projeto de código aberto ajuda muito, pois milhares de programadores de todo mundo estão nesse momento analisando os códigos-fonte dele em busca de melhorias. Outro fato é que usuários de código aberto já estão acostumados a cooperarem entre si, prestando suporte gratuito e eficiente.

A maioria do suporte é feita em fóruns ou grupos de e-mail, mantidos por voluntários. Isso requer alguns cuidados quando for solicitar ajuda, como por exemplo:

- **Seja gentil:** Apresente-se, respeite as regras do Grupo. Lembre-se que todos os usuários do grupo são voluntários;
- **Seja específico:** Detalhe bem o seu problema, e o ambiente onde ele ocorre. Para que os interessados em ajudá-lo consigam compreendê-lo corretamente;
- **Pesquise antes**: Ninguém gosta repetir várias vezes a mesma resposta. Provavelmente a sua dúvida já foi respondida em

Extensão	Finalidade	Observações	Delphi
*.LPI	Arquivo com as Informações do projeto.	Armazenado em XML, contém as configurações específicas do projeto. Utilize-o sempre para abrir o projeto, ao invés do arquivo associado *.lpr	*.DPR
*.LPR	Arquivo de projeto do Lazarus.	Contém o código-fonte do programa principal.	*.DPR
*.LFM	Definições do Formulário.	Contém informações da configuração de todos os objetos do formulário. Sempre está associado a um arquivo *.pas correspondente. Armazenado em formato texto, muito semelhante ao .dfm do Delphi em modo texto (compatível, em muitos aspectos).	*.DFM
*.PAS *.PP	Unit com código-fonte em Pascal	Normalmente está associado a um arquivo de formulário correspondente (*.lfm)	*.PAS
*.LRS	Arquivo de resource do Lazarus.	Normalmente, esse tipo de arquivo é gerado pelo Lazarus durante a compilação. Este tipo pode ser criado pelo programa LazRes, disponível em "c:\lazarus\tools" (é necessário compilá-lo). Não deve ser confundido com o arquivo de Resource do Windows (não é compatível). Antes da versão 0.9.29 o Lazarus não era compatível com arquivos de Resource do Windows. Portanto, compilava os arquivos .lfm em .lrs. Porém, atualmente o Lazarus pode usar diretamente os arquivos .lfm como Resources, assim como o Delphi.	N/A
*.PPU	Unit compilada.	Arquivo gerado pelo compilador FPC, para cada Unit (*.pas; *.pp), durante o processo de compilação.	*.DCU
*.LPK	Arquivo com as informações de pacote (Package)	Armazenado em XML. Contém configurações específicas do pacote.	*.DPK

Tabela 1. Descrição dos sufixos de arquivos

outra ocasião. Então use o mecanismo de busca do Fórum (ou o Google), e verifique. Se não encontrar nada que lhe ajude, então pergunte;

• **Seja grato**: Retorne para falar se o problema foi resolvido ou não. Isso incentiva (muito) os voluntários continuarem com o trabalho;

• **No Brasil:** Faça a sua inscrição no grupo de e-mail do Lazarus-BR. Lá você entrará em contato com vários programadores que usam Lazarus há bastante tempo, e que colaboram efetivamente no aprimoramento do mesmo;

• **Canais oficiais (em inglês):** As listas de e-mail oficiais são bastante movimentadas, ou seja, você receberá uma grande quantidade de e-mails diariamente ao se inscrever nelas. Se você quer entrar em contato direto com os desenvolvedores do Lazarus, este é o lugar certo. O idioma obrigatório nessas listas é o Inglês. Os usuários da lista compreendem que nem todos são fluentes na língua inglesa, e são bastante tolerantes. Portanto, mesmo contando com a ajuda de uma ferramenta de tradução, não hesite em usar a lista;

• **Suporte pago:** O suporte pode ser contratado por hora, dia, mês ou ano. A empresa responsável por esse site é intimamente ligada ao desenvolvimento do Lazarus.

Como ajudar no projeto?

O Lazarus, como todo projeto de código aberto, só terá sucesso, se puder contar com a participação ativa de sua comunidade de usuários. A seguir forneço algumas dicas de como você pode se envolver ativamente no projeto. Mesmo que você não possua um grande conhecimento de Object Pascal, ainda assim poderá ajudar.

Escrevendo relatórios de erros

Devido o fato do Lazarus ser um projeto em constante evolução, é possível que algum problema ou efeito colateral seja introduzido acidentalmente, devido a alguma modificação recente. Se você se deparar com uma situação de erro no Lazarus, não deixe de entrar em contato com as listas de suporte. Os desenvolvedores do Lazarus dão prioridade na correção de erros graves ou que impeçam a compilação do IDE. Se você tiver certeza que o problema é realmente no Lazarus, você pode postar um relatório de erro no BugTracker do Lazarus. O BugTracker do Lazarus é gerenciado pelo programa MantisBT, que é muito simples de usar. É necessário ter uma conta para relatar erros. Após efetuado o login, clique em "Relatar Erro", escolha o projeto em que mais se aplica o erro, e siga as instruções na tela. Os relatórios de erro sempre devem ser feitos em Inglês. Alguns pontos importantes no processo de criação de um relatório de erro:

• **Pesquise se o erro já foi reportado:** Verifique se alguém já relatou o problema. Use o mecanismo de filtro do MantisBT com palavras chaves adequadas. Você pode monitorar casos do seu interesse. Apenas clique no botão "Monitorar", quando o caso estiver aberto (é necessário estar logado);

• **Tenha certeza que o problema é no Lazarus**: Às vezes o problema pode ser algo nos fontes do seu projeto. Procure isolar o problema. Se for o caso, faça um miniprojeto bem simples. Por exemplo: Crie um Form com apenas um botão, que quando clicado execute uma rotina que seja capaz de reproduzir o erro. Compacte apenas os fontes desse projeto, e anexe-o no relatório de erros;

• **Crie um roteiro de reprodução do problema:** Se você não possui um miniprojeto que demonstre o erro, então informe exatamente quais são os passos necessários para reproduzi-lo. Um roteiro passo a passo, com uma operação por linha;

• **Identifique o seu ambiente de trabalho:** É comum um erro acontecer apenas em uma situação muito específica. Portanto, detalhe o seu Sistema Operacional, versão do Lazarus, versão do FPC e qual WidgetSet você está utilizando;

Enviando correções

Sem dúvida isso requer um conhecimento mais avançado dos fontes do Lazarus. Outro requisito é estar apto para baixar os fontes do Lazarus pelo SVN, pois as alterações devem ser feitas com base na última versão do Lazarus (trunk).

Antes de enviar suas correções, questione nas listas oficiais se a sua ideia está correta, principalmente quando se tratar da sua primeira sugestão. Uma vez que você testou sua correção e verificou que ela está funcionando corretamente, você deve gerar um "patch" ("remendo") dos arquivos alterados, através de algum cliente de SVN, como por exemplo, o TortoiseSVN.

Reporte o caso no BugTracker do Lazarus, da mesma maneira que você reportaria um erro, porém escolha o projeto "Lazarus>Patches". Forneça informações detalhadas sobre o problema encontrado, como ele pode ser reproduzido, e qual foi a sua estratégia de solução para o mesmo. Por último, anexe o arquivo de patch no seu relatório de erro.

Os desenvolvedores do Lazarus irão analisar os fontes do seu patch. Caberá a eles decidir aplicar ou não o seu patch contra os fontes originais do SVN. No caso de recusa, sempre haverá uma explicação do motivo. Após você se tornar um colaborador frequente, enviando vários patches de boa qualidade, provavelmente você será convidado para compor seu nome na lista de colaboradores do Lazarus, que pode ser acessada pelo IDE em: Ajuda>Sobre>Colaboradores.

Efetuando doações

Atualmente, a equipe de desenvolvimento do Lazarus é composta exclusivamente por voluntários. Não há desenvolvedores pagos como ocorre em alguns projetos de código aberto, como por exemplo: Mozilla e Firebird. Provavelmente isso ocorre pelo fato do Lazarus ainda não ser uma fundação ou algo semelhante. Entretanto, o projeto tem algumas despesas, como por exemplo, manter em funcionamento os servidores que hospedam todos os serviços: BugTracker, SVN, FTP, WebServer etc. Caso deseje mostrar sua gratidão ao projeto, colaborando com uma doação, utilize o botão "Donate" que pode ser encontrado na página principal do projeto.

Outra maneira de ajudar no projeto são as recompensas pagas. Suponha que você queira muito um novo recurso ou correção no Lazarus e esteja disposto a pagar por isso. Nesse caso, você pode oferecer uma

recompensa monetária para que essa tarefa seja executada. Cabe a você, detalhar minuciosamente a tarefa e o objetivo esperado, além de definir o prêmio. Caso seja do interesse de algum desenvolvedor do Lazarus, ele pode efetuar o trabalho e receber a recompensa financeira e as melhorias feitas por ele são incorporadas nos fontes do projeto original. Dessa maneira, você efetivamente contribuiu com melhorias nos fontes do projeto, mesmo que apenas pagando por elas. Para mais informações sobre como criar novas recompensas e para ver as recompensas em aberto acesse a página correspondente na sessão Links.

Conclusão

Nestes dois artigos pudemos abrir espaço para um IDE completo, Open-Source de excelente qualidade. Você que gosta de projetos Open-Source e linguagem Object Pascal, não pode deixar de conhecer este projeto. Vimos como instalar, configurar, customizações, otimizações, instalação de componentes e muitos outros recursos. Espero que a comunidade tenha despertado interesse pela ferramenta e que todos tenham gostado.

Daniel Simões Almeida
daniel@djsystem.com.br
Sócio/proprietário da empresa DJSystem Consultoria e Automação Comercial. Idealizador e principal mantenedor do Projeto ACBr (Automação Comercial Brasil). Graduado em Processamento de Dados e pós-graduado em Análise de Sistemas pela universidade UNIMEP (Universidade Metodista de Piracicaba). É responsável pelo setor de tecnologia e desenvolvimento da DJSystem, programando em ObjectPascal, xBase e PHP, nas plataformas Windows e Linux.

Wiki Lazarus
*http://wiki.lazarus.freepascal.org/*Lazarus_For_Delphi_Users
http://wiki.lazarus.freepascal.org/Code_Conversion_Guide

Blog do colega Sílvio Clécio
http://silvioprog.blogspot.com/

Blog do Professor Carlos Araújo
http://professorcarlos.blogspot.com/

Blog do colega Luiz Américo
http://lazarusroad.blogspot.com/

Blog dos desenvolvedores do Lazarus
http://lazarus-dev.blogspot.com/

Download TortoiseSVN
http://tortoisesvn.net

Página do CCR
http://lazarus-ccr.sourceforge.net/

Página com mais informações do CCR
http://wiki.lazarus.freepascal.org/Components_and_Code_examples

Endereço para download dos componentes compactados
http://sourceforge.net/projects/lazarus-ccr/files/

Componente ZeosLib
http://zeos.firmos.at/

Componente Projeto ACBr
http://acbr.sourceforge.net/

Componente Fortes Report
https://fortes4lazarus.svn.sourceforge.net/svnroot/fortes4lazarus/trunk

Componente ExtPascal
http://code.google.com/p/extpascal/

ExtPascal (Demos)
http://extpascal.call.inf.br/cgi-bin/extpascalsamples.cgi

Componente Fast Report para Lazarus (LazReport)
http://www.fast-report.com/pt/download/free-report-download.html

Componente Synapse
http://www.ararat.cz/synapse

Conheça mais sobre Widgets
http://wiki.lazarus.freepascal.org/Roadmap

Grupo brasileiro de Lazarus
http://groups.google.com.br/group/lazarus-br

Lista oficial de Lazarus
http://lists.lazarus.freepascal.org/mailman/listinfo

Fórum oficial
http://www.lazarus.freepascal.org/index.php?action=forum

Suporte pago
http://www.lazarussupport.com/

BugTracker do Lazarus
http://bugs.freepascal.org/

Página de recompensas
http://wiki.lazarus.freepascal.org/Bounties

Dê seu feedback sobre esta edição!

A Clubedelphi tem que ser feita ao seu gosto. Para isso, precisamos saber o que você, leitor, acha da revista!

Dê seu voto sobre este artigo, através do link:
www.devmedia.com.br/clubedelphi/feedback

Design Patterns –

Abstract Factory e Factory Methods

Como se sabe a orientação a objetos sempre foi um dos pilares das boas práticas de programação, permitindo que seja possível construir aplicações robustas e ao mesmo tempo flexíveis, ou seja, que sejam capazes de sofrer modificações sem tornar isso uma tarefa árdua e desgastante. Embora todo o IDE do Delphi seja Orientado a Objetos, esta ainda é uma prática pouco utilizada pelos seguidores do Object Pascal, devido à metodologia procedural que sempre foi adotada pela maioria. Porém, acompanhando fóruns de discussão, é possível notar que este quadro tem se revertido, mostrando um interesse maior e uma mudança de ideias em relação à adoção da orientação a objetos como metodologia cotidiana dos desenvolvedores Delphi.

Para a melhor utilização da metodologia é necessário compreender os principais fundamentos e conceitos, e porque não dizer obrigatórios e indispensáveis, para visualizar a orientação a objetos de forma clara e organizada. Para isto, foram criados os padrões de projeto, que ganharam popularidade através dos autores Erich Gamma, Richard Helm, Ralph Johnson e John Vlissides, ou GoF (Gang Of Four) como é mais conhecida, através do livro Design Patterns: Elements of Resuble Object-Oriented Software.

No artigo anterior foi comentado a respeito do padrão de projeto Factory Method. Factory Method é um método que se responsabiliza pela criação (e possível configuração ou ligação) de um objeto ou de um grupo de objetos.

Nesse contexto o construtor de uma classe (Create no Delphi Win32, new no Delphi Prism) pode ser considerado como um método fábrica degenerado, pois cria apenas um tipo de objeto, sem parametrizá-lo muito e ainda tem um alto acoplamento com a unidade desta classe por motivos óbvios: ele foi escrito como método desta classe, nesta mesma unidade.

Já nesta parte da série, será abordado o Abstract Factory. Para facilitar a compreensão do padrão, imagine já ter criado melhorias em um sistema por delegar a criação de certos objetos a alguns métodos que serão usados por "clientes" (outros objetos que necessitem destes).

A limitação aqui refere-se a necessidade de instanciar uma classe fábrica concreta para cada tipo ou família de objetos que seja criada. Supondo-se por exemplo, que uma fábrica de conexões com o Interbase fosse criada, ela criaria tanto o IBConnection como os IBDataSets, e talvez uma outra fábrica concreta poderia criar AdoConnections e AdoDataSets.

O objetivo do Abstract Factory é criar uma interface comum para acesso e manipulação de qualquer fábrica concreta.

Resumo DevMan

De que trata o artigo:

Neste artigo continuaremos o estudo sobre os padrões de projeto. Iremos abordar o padrão Abstract Factory, implementando-o de duas formas: como uma classe abstrata e como uma interface. Daremos um enfoque maior à segunda opção.

Para que serve:

Abstract Factory não é apenas a ancestral de todas as Concrete Factorys, mas é também um modo de desacoplar ainda mais os seus sistemas, por delegar a criação de famílias inteiras de objetos sem se especificar uma classe concreta e usando uma única interface. Pode ser até uma fábrica de fábricas.

Em que situação o tema é útil:

Quando o seu sistema necessita de um alto nível de generalização ele também precisará de um alto nível de desacoplamento. Enquanto as concrete Factorys permitem criar uma gama diferente de objetos da mesma família através de seus Factory Methods, contanto que a fábrica concreta seja especificada, Abstract Factory pode flexibilizar até mesmo qual será a fábrica concreta a ser usada, delegando o tipo de retorno de seus Factory Methods a seus descendentes. Isso nos permitirá, por exemplo, criar conexões com bancos de dados diferentes por usar linhas/fabricantes de componentes diferentes.

Design Patterns:

Desacoplar é a chave para flexibilizar. Se o seu sistema for capaz de montar automaticamente todo um fluxo que vai desde as interfaces, passando pelos objetos de negócio e indo até a persistência, livre de qualquer dependência com outras tecnologias podemos dizer que o sistema é flexível. A flexibilidade no Delphi ainda traz os benefícios de se estar usando uma linguagem compilada, nativa e fortemente tipada.

Nota do DevMan

Os padrões de projeto são basicamente divididos em três famílias:

Padrões de criação ou criacionais: Abstract Factory, Factory Method, Singleton, Builder e Prototype;

Padrões estruturais: Adapter, Bridge, Composite, Decorator, Façade, Flyweight e Proxy;

Padrões comportamentais:Chain of Responsibility, Command, Interpreter, Iterator, Mediator, Memento, Observer, State, Strategy, Template Method e Visitor

Parte 2

VITOR LUIZ RUBIO

Nota do DevMan

Interfaces, contagem de referência, TInterfacedObject e IInterface

Objetos que implementam interfaces, caso sejam descendentes de TInterfacedObject não necessitam ser destruídos, pois tem um contador de referências interno que automaticamente destrói o objeto que a implementa quando esse contador chega a zero. Além disso, TInterfacedObject tem um mecanismo para trabalhar com multithread que previne que um objeto seja destruído quando ainda não terminou de ser criado. Então, teoricamente usando interfaces e criando classes como descendentes de TInterfacedObject não precisamos nos preocupar com a destruição dos objetos e nem com Memory Leaks, que é o termo dado para um objeto que foi criado em tempo de execução, contudo, após seu devido uso e descarte, não é destruído / eliminado da memória.

Como sabemos, toda classe em Object Pascal acabará herdando de uma outra conhecida como TObject. Desse modo, os métodos dessa classe estarão disponíveis a qualquer outra que for herdada. Não surpreendentemente, o mesmo acontece com as interfaces. Toda e qualquer interface no Delphi, mesmo que não seja declarado explicitamente, herdará de IInterface.

A diferença na herança entre classes e interfaces está na falta de implementação. Se uma interface X herda de Y, um objeto que implemente X também deverá implementar os métodos de Y. Ou seja, se seguirmos essa ideia até o topo da hierarquia, todos os objetos que implementem interfaces, deverão definir implementações para os métodos de IInterface. Na prática normalmente não implementamos diretamente nenhum método de IInterface (mesmo que ainda não saibamos quais eles sejam).

Veja a seguir a declaração de IInterface e TInterfacedObject, retirada dos fontes da VCL do Delphi XE:

```
IInterface = interface
  ['{00000000-0000-0000-C000-000000000046}']
  function QueryInterface(const IID: TGUID; out Obj): HResult; stdcall;
  function _AddRef: Integer; stdcall;
  function _Release: Integer; stdcall;
end;

TInterfacedObject = class(TObject, IInterface)
protected
  FRefCount: Integer;
  function QueryInterface(const IID: TGUID; out Obj): HResult; stdcall;
  function _AddRef: Integer; stdcall;
  function _Release: Integer; stdcall;
public
  procedure AfterConstruction; override;
  procedure BeforeDestruction; override;
  class function NewInstance: TObject; override;
  property RefCount: Integer read FRefCount;
end;
```

Perceba que TInterfacedObject já faz o trabalho de declarar e codificar os métodos de IInterface. Sendo assim, tais métodos já foram implementados para qualquer classe que a herde, o que nos livra do trabalho da implementação a cada nova declaração. É altamente aconselhável que sempre usemos essa classe para implementar interfaces.

Mas o que há de tão especial em IInterface? Observamos um método para consulta e dois para a contagem de referências. Um tópico muito importante é que essa contagem inibe a necessidade de executarmos um destrutor para esses objetos. Ou seja, nunca precisaremos chamar o método Free de um objeto que utiliza uma interface.

Isso funciona da seguinte maneira: a cada nova referência ao objeto, o método _AddRef é chamado, o que incrementa o valor de RefCount de um TInterfacedObject. Da mesma forma, a cada perda de referência, o método _Release é executado e a contagem é decrementada. Quando o contador chegar a 0, o objeto é liberado da memória. Veja um exemplo:

```
Objeto1 := TClasseExemplo.Create();
{ Referências ao objeto criado: 1 }
Objeto2 := Objeto1;
{ Referências ao mesmo objeto: 2 }
Objeto2.Metodo();
Objeto1 := nil;
{ Perdeu uma referência, total: 1 }
Objeto2 := nil ;
{ Nenhuma referência ao objeto. Retirado da memória }
```

Com o Abstract Factory criam-se variáveis genéricas, com métodos que criarão o objeto indiretamente, delegando aos descendentes da Abstract Factory a responsabilidade de saber qual o tipo de objeto correto a se criar.

Há duas formas de atribuir uma interface comum a todas as fábricas de objetos: ou elas são descendentes do mesmo ancestral comum ou elas implementam uma mesma interface.

Será adotada a segunda opção pelas vantagens que o uso de interfaces proporciona: contagem de referências (veja nota do DevMan) e a possibilidade de criar uma fábrica concreta que não seja necessariamente "da família" das outras fábricas. Porém, antes de criar uma interface padrão para as fábricas concretas, será criada uma interface padrão para os produtos concretos. Isso flexibilizará a aplicação a tal ponto que para um objeto ser considerado um produto concreto basta que implemente a interface, mesmo que não seja "parente" dos outros produtos. Isso exibe um cenário em que virtualmente qualquer classe pode ser uma fábrica e criar qualquer tipo de produto.

Pense na paleta de componentes do Delphi. Os componentes são apenas classes normais que descendem da classe TComponent, implementando certas fun-

cionalidades para que o IDE do Delphi possa conhecê-las, interagir com elas e executar seu código em Design Time. Isso quer dizer que objetos da classe TEdit são criados quando se coloca um Edit em um formulário, e são destruídos quando é pressionada a tecla "delete". Em Design Time os construtores e destrutores dos objetos são executados, portanto se existir um "ShowMessage('olá mundo');" em um construtor de um componente, ele será executado quando o componente for colocado no Form.

Então isso faz do IDE do Delphi, principalmente da paleta de componentes, uma gigantesca fábrica, que cria os componentes através de seus nomes de classe. No artigo também será abordada uma forma de como criar um objeto de uma classe a partir de seu nome.

Listagem 1. O produto abastrato poderia ser uma classe assim

```
TAbstractProduct = class
public
  function ShowWhoIAm: string; virtual;
   abstract;
  function GetNome: string; virtual; abstract;
  function GetDocumento: string; virtual;
   abstract;
  procedure SetNome(valor:string); virtual;
   abstract;
  procedure SetDocumento(valor:string);
   virtual; abstract;
  property Nome: string read GetNome write
   SetNome;
  property Documento: string read
   GetDocumento write SetDocumento;
end;
```

Listagem 2. Uma interface comum para todos os produtos concretos

```
unit uAbstractProduct;
interface
type
  IAbstractProduct = interface
  ['{6E4392A0-1CC0-4B9A-BEE2-BB9FE54B5A20}']
    function ShowWhoIAm: string;
    function GetNome: string;
    function GetDocumento: string;
    procedure SetNome(valor:string);
    procedure SetDocumento(valor:string);
    property Nome: string
        read GetNome write SetNome;
    property Documento: string
        read GetDocumento write SetDocumento;
  end;
implementation
end.
```

Nota do DevMan

Uma curiosidade, até o Delphi 5, a interface base de todas as interfaces do Delphi não era IInterface, mas sim IUnknown. Essa por sua vez tinha sua definição fortemente dependente do modelo de objetos da Microsoft, chamado na época de COM (Component Object Model). Com o surgimento do Kylix 1, antes do Delphi 6, a VCL foi portada para ser compatível com a plataforma Linux (originando assim a CLX), onde não existe o COM. Dessa forma, foi definida uma nova interface, que no mundo Windows apareceu a partir do Delphi 6 e permanece até hoje. Nascia IInterface. Porém, imagine quantas classes no mundo já não apontavam para esta interface ou se ligavam a outras por meio de IUnknown. Mudar a declaração de todas iria ferir um dos princípios mais básicos da orientação a objetos. Dessa forma, a equipe do Delphi resolveu o problema de uma forma extremamente simples, usando um alias, como você pode observar nos fontes da própria VCL:

```
IUnknown = IInterface;
```

Nota do DevMan

Algumas dicas para entender os padrões Factory Methods e Abstract Fectory:

- Todas as fábricas encapsulam a criação de objetos;

- Factory Methods utiliza a herança: a criação dos objetos é realizada pelas classes filhas que implementam os métodos da fábrica para a criação dos objetos;

- Abstract Factory utiliza a composição de objetos: a criação de objetos é implementada em métodos expostos na interface de fábrica;

- Todos os padrões Factory promovem ligações fracas reduzindo a dependência de seu aplicativo em classes concretas;

- O objetivo do Factory Method é permitir que uma classe defira a criação de instâncias para suas subclasses;

- O objetivo da Abstract Factory é criar famílias de objetos relacionados sem ter que depender de suas subclasses;

- Factorys são uma técnica eficaz para codificar para abstrações, e não classes concretas;

Métodos de fábrica com interfaces

No artigo anterior foi criada uma classe para produto abstrato chamada TAbstractProduct, da qual descendiam as classes TPessoaFisica e TPessoaJuridica, e também um outro tipo de Abstract Product que eram os formulários de cadastro: criamos um formulário "abstrato" para ser ancestral dos dois formulários (concretos) programados para interagir com os outros objetos concretos. Porém até então não havia uma fábrica abstrata, pois a fábrica que foi feita era concreta.

O produto abstrato a que chegamos no artigo anterior poderia ser implementado como o da **Listagem 1**, com os métodos e propriedades comuns a todos os produtos. Em vez disso, uma interface em comum será extraída dos produtos concretos. Isso quer dizer que os produtos concretos não mais precisarão descender de um mesmo ancestral, bastando que implementem a mesma interface.

Criar uma interface única flexibiliza muito o projeto, porque dá a qualquer classe o potencial para se tornar um objeto concreto, bastando que implemente a interface da **Listagem 2**, mesmo que esses objetos concretos não tenham nenhum grau de parentesco entre si.

Você deve pensar de forma "abstrata". Da mesma forma que um construtor é um método fábrica degenerado porque cria um único tipo de objeto e não inicializado, pode-se também imaginar que uma interface é uma classe abstrata degenerada, mas com propriedades especiais.

Embora a interface não agregue nada a seus "descendentes", ela força a implementação de certos métodos, fazendo uma espécie de "contrato". Qualquer classe que seguir esse contrato está apta a ser um substituto para a interface. Pode-se também afirmar que qualquer classe tem uma interface "implícita", sendo caracterizada pelo conjunto de suas propriedades e métodos. Quando extraímos um "denominador comum"

Nota

Se você estiver usando uma das novas versões do Delphi do 2006 ao XE, que suporta classes abstratas, poderá colocar a palavra chave "abstract" depois da declaração da classe.

de várias classes essa passa a ser uma interface "explícita".

No exemplo notamos que nossos dois objetos concretos possuíam algo como um nome e um documento. Por isso extraímos de ambos a interface da **Listagem 2** e agora, nas **Listagens 3** e **4** teremos que modificar as classes de produto concreto para que implementem a interface IAbstractProduct e descendam de TInterfacedObject.

O form básico FrmCadastroAbstrato agora tem o método CarregaCliente recebendo um objeto do tipo IAbstractProduct (**Listagem 5**). A única mudança necessária foi alterar o TAbstractProduct por IAbstractProduct. Perceba na **Listagem 6** que os formulários TFrmCadastroPF e TFrmCadastroPJ também foram modificados.

Com isso eliminamos a necessidade de Type Casting nesses formulários e também a necessidade de colocar as units uTPessoaFisica e uTPessoaJuridica no uses dos formulários. Esses pequenos ajustes foram suficientes, mas é possível evoluir ainda mais este exemplo.

Foi criado um acoplamento desnecessário entre o formulário principal e a classe fábrica. Não existe motivo para esse acoplamento, e ele poderia atrapalhar caso houvesse uma família maior de objetos, fábricas concretas e fábricas abstratas.

ConcreteFactorys deveriam ser criadas na inicialização do programa e destruídas na sua finalização, implementadas como um objeto Singleton para serem chamadas onde e quando precisar, preferencialmente em uma unit separada.

A ConcreteFactory está agora criando objetos de duas famílias diferentes, a família das pessoas físicas e a família das pessoas jurídicas. Possui um acoplamento desnecessário com a unit IniFiles e ainda tem um código repetitivo, estilo copiar – colar, em dois métodos diferentes, coisa que deveria ser abominada por quem programa orientado a objetos.

Seria interessante se cada ConcreteFactory criasse objetos de apenas uma família, compatíveis entre si, não importa quantos Factory Methods ela tenha. Para estes casos utiliza-se o padrão Abstract Factory, criando uma fábrica que terá como responsabilidade adiar para suas subclasses ao tipo correto de objetos a serem criados. Isso significa que uma fábrica não irá construir dois tipos de objetos, mas apenas um, porém, de forma transparente.

Se cada fábrica será especializada em um tipo de família de objeto, deverá instanciar a fábrica correta, e na hora de decidir qual ConcreteFactory criar, deverá usar um Factory Method, como mostra a **Listagem 7** (que define nossa Abstract Factory).

Esta unit é bem simples, mas veja como as coisas se encaixam: a fábrica abstrata está definida como uma classe que tem um método CreateProduct que retorna um IAbastractProduct, e CreateInterface que retorna um objeto do tipo TFrmCadastroAbstrato. Isso significa que um descendente de TAbstractFactory, por exemplo TFabricaDeTijolos retornaria um TTijolo com o método CreateProduct, e um formulário para edição de tijolos com o CreateInterface.

Dessa forma, no exemplo apresentado, teremos duas fábricas descendentes. A **Listagem 8** mostra a fábrica de pessoa física e a **Listagem 9** a fábrica de pessoa jurídica.

A fábrica de pessoas físicas possui acoplamentos apenas com as units relacionadas a pessoas físicas. Como foi adotada uma interface comum entre ambos os objetos concretos, não será necessário fazer Type Casting na hora de preencher as propriedades do objeto-exemplo com valores. Como você pôde notar, o mesmo ocorreu com pessoas jurídicas.

Pode-se decidir qual fábrica concreta criar no começo da aplicação e usar sempre a mesma, sob demanda. Outra opção seria criar todas as fábricas concretas e decidir qual usar na hora de solicitar a um Factory Method a criação de um objeto.

A **Listagem 10** mostra uma unit que contém uma classe responsável por decidir

Listagem 3. TPessoaFisica

```
unit uTPessoaFisica;
...
 TPessoaFisica = class(TInterfacedObject,
   IAbstractProduct)
...
```

Listagem 4. TPessoaJuridica

```
unit uTPessoaJuridica;
...
 TPessoaJuridica = class(TInterfacedObject,
IAbstractProduct)
...
```

Listagem 5. Formulário ancestral

```
unit uFrmCadastroAbstrato;
...
 TFrmCadastroAbstrato =  class(TForm)
   procedure CarregaCliente(
     cliente: IAbstractProduct); virtual; abstract;
```

Listagem 6. Adaptando os formulários descendentes

```
unit uFrmCadastroPF;
...
 TFrmCadastroPF = class(TFrmCadastroAbstrato)
...
   procedure CarregaCliente(
     cliente: IAbstractProduct); override;
...
unit uFrmCadastroPJ;
...
 TFrmCadastroPJ = class(TFrmCadastroAbstrato)
...
   procedure CarregaCliente(
     cliente: IAbstractProduct); override;
...
```

Nota do DevMan

Ao implementar IAbstractProduct você deve fazer com que sua classe, antes um descendente de TAbstractFactory, descenda de TInterfacedObject. Isso porque TInterfacedObject implementa os métodos básicos de IInterface, comuns a qualquer interface e que são responsáveis pelo gerenciamento de memória e contagem de referências das interfaces (como comentou o DevMan em sua nota anterior) e por algumas das funções da RTTI sobre interfaces.

Nota do DevMan

Singleton é um padrão de projeto que permite criar uma classe que pode ser instanciada apenas uma vez. Toda tentativa de se acessar ou criar um objeto desses retornará a mesma instância do objeto.

Nota do DevMan

Diferentemente de uma interface, uma classe abstrata não precisa ter todos os seus métodos implementados ou sobrescritos por algum descendente direto (filho), podendo delegar essa tarefa a um descendente de seu descendente. Porém chamar um método abstrato ou ainda não implementado de uma classe qualquer gerará um erro.

Listagem 7. Fábrica Abstrata

```
unit uAbstractFactory;

//eu não tenho conhecimento dos concretos

interface

uses
 uAbstractProduct,
 uFrmCadastroAbstrato;

type
 TAbstractFactory = class
 public
  function CreateProduct:IAbstractProduct; virtual; abstract;
  function CreateInterface: TFrmCadastroAbstrato; virtual; abstract;
 end;

implementation

end.
```

qual fábrica concreta criar. Ela tem uma propriedade de classe do tipo TAbstractFactory que será um Singleton da fábrica concreta. Existem várias formas de se implementar um Singleton (que também é um padrão de projeto) em Delphi, mas optou-se pela forma mais simples, que implementa o Singleton baseado num property Field.

Na **Listagem 10** foram usados Class Methods, ou métodos de classe. Esses métodos são estáticos e são acessíveis a partir do nome da classe. Podem ser criados utilizando a palavra reservada class antes da procedure ou function. Isto foi feito para garantir que a classe TFactorySingleton possa ser usada sem ser criada, afinal ela só servirá para criar as fábricas concretas.

Não faria sentido nenhum criar um TFactorySingleton e nem há porque fazer isso. TFactorySingleton não é um Singleton. Na verdade ele faz o controle para que a fábrica concreta seja um, encapsulando sua criação e disponibilizando uma maneira restrita e padrão para acessá-la.

Para garantir que haja uma única instância, foi usada uma propriedade de classe, que nada mais é do que uma propriedade estática que

Listagem 8. Fábrica de Pessoa Física

```
unit uPessoaFisicaFactory;

interface

uses
 uAbstractFactory,
 uAbstractProduct,
  uFrmCadastroAbstrato,
 uTPessoaFisica,
 uFrmCadastroPF;

type
 TPessoaFisicaFactory = class(TAbstractFactory)
 public
  function CreateProduct:IAbstractProduct; override;
  function CreateInterface: TFrmCadastroAbstrato; override;
 end;

implementation

{ TPessoaFisicaFactory }

function TPessoaFisicaFactory.CreateInterface: TFrmCadastroAbstrato;
begin
 Result := TFrmCadastroPF.Create(nil);
end;

function TPessoaFisicaFactory.CreateProduct: IAbstractProduct;
begin
 Result := TPessoaFisica.Create;
 Result.Nome := 'Vitor Luiz Rubio';
 Result.Documento := '11111111111';
end;

end.
```

Listagem 9. Fábrica de Pessoa Jurídica

```
unit uPessoaJuridicaFactory;

interface

uses
 uAbstractFactory,
 uAbstractProduct,
 uFrmCadastroAbstrato,
 uTPessoaJuridica,
 uFrmCadastroPJ;

type
 TPessoaJuridicaFactory = class(TAbstractFactory)
 public
  function CreateProduct:IAbstractProduct; override;
  function CreateInterface: TFrmCadastroAbstrato; override;
 end;

implementation

{ TPessoaJuridicaFactory }

function TPessoaJuridicaFactory.CreateInterface: TFrmCadastroAbstrato;
begin
 Result := TFrmCadastroPJ.Create(nil);
end;

function TPessoaJuridicaFactory.CreateProduct: IAbstractProduct;
begin
 Result := TPessoaJuridica.Create;
 Result.Nome := 'Rubisoft Software';
 Result.Documento := '11111111000111';
end;

end.
```

permite ser criada em uma classe usando a palavra reservada var. A variável foi criada para ser um campo privado e estático. Juntamente foi definido um método de classe CreateFactory para criar a fábrica e GetFactory para obter o valor. Com isso usamos o encapsulamento para preservar a instância da fábrica (FFactory) de ser destruída ou substituída. Para garantir que apenas uma FFactory existirá, esta foi definida como estática, como se fosse uma variável pública global, mas ela não é pública e muito menos global.

O método CreateFactory retorna uma instância de uma das duas fábricas concretas e é ele que encapsula toda aquela lógica de descobrir qual fábrica criar através de um arquivo ini, conforme foi visto. Como esse método de classe é privado, ele poderá ser executado só por outro método da própria classe. É aí que entra o código do método GetFactory.

Isso se chama Lazy Binding. A propriedade Factory sempre estará disponível para quem quiser usar, mas a primeira vez que for usada ela chama GetFactory que por sua vez chama CreateFactory, armazenando a instância em FFactory. Se FFactory já estiver criada então ela simplesmente a retorna, sem fazer mais nada. Isso garante que haverá apenas uma instância de fábrica, simulando assim o padrão Singleton.

Nada impede que se possa instanciar vários objetos TPessoaFisicaFactory ou TPessoaJuridicaFactory usando o método construtor Create dessas classes. Isso exigiria apenas acesso as suas units. Por enquanto será adotada sempre a propriedade Factory da classe TFactorySingleton. Repare que foi definida nessa unit uma sessão de finalização para liberar o único objeto Concrete Factory da memória.

Agora simplesmente mudar o arquivo INI em runtime não causará efeito algum, a não ser que a aplicação seja reiniciada ou o objeto Factory liberado, para que outro seja criado. Porém liberar o Singleton da

Nota

Existem maneiras de se fazer um Singleton que não permitem criação de uma segunda instância nem pelo método Create, mas isso foge do escopo desse artigo.

Listagem 10. Singleton para criação de fábricas

```
unit uFactorySingleton;

//eu não tenho conhecimento nem dos produtos
//abstratos, nem dos concretos

interface

uses
 uAbstractFactory,
 uPessoaFisicaFactory,
 uPessoaJuridicaFactory;

type
 TFactorySingleton = class
 private
    //class property a ser usada em todo o sistema
    //que contém a factory correta
  class var FFactory: TAbstractFactory;
  //função que irá criar a factory efetivamente
  class function CreateFactory: TAbstractFactory;
  class function  GetFactory: TAbstractFactory; static;
 public
  class procedure LiberarFactory;
  class property Factory: TAbstractFactory read GetFactory;
 end;

implementation

uses
 inifiles,
 forms;

{ TFactorySingleton }

class function TFactorySingleton.CreateFactory: TAbstractFactory;
var
 TypeProduct: string;
begin
 with TIniFile.Create(Application.ExeName+'.ini') do
 try
  TypeProduct := ReadString('Aplicacao', 'TipoEmpresa', '');
  if TypeProduct = 'Atacado' then
  begin
    Result := TPessoaJuridicaFactory.Create;
  end
  else
   if TypeProduct = 'Varejo' then
  begin
   Result := TPessoaFisicaFactory.Create;
  end
  else
   Result := nil;
 finally
  free;
 end;
end;

class function TFactorySingleton.GetFactory: TAbstractFactory;
begin
 //esse padrão é bom porque,
 //embora a propriedade esteja sempre acessível,
 //ela será criada somente uma vez e sob demanda
 if FFactory = nil then
 begin
  FFactory := CreateFactory;
 end;
 Result := FFactory;
end;

class procedure TFactorySingleton.LiberarFactory;
begin
 FFactory.Free;
 FFactory := nil;
end;

initialization

finalization
 //liberar da memória a única instância do
 //objeto Concrete Factory (singleton)
 TFactorySingleton.LiberarFactory;
end.
```

Listagem 11. Unit principal reformulada

```
unit Unit1;

interface

uses
...
 uFactorySingleton,
...
type
 TForm1 = class(TForm)
...
   procedure Button1Click(Sender: TObject);
...
procedure TForm1.Button1Click(Sender: TObject);
begin
 with TFactorySingleton.Factory.CreateInterface do
 begin
  CarregaCliente(TFactorySingleton.Factory.
   CreateProduct);
  ShowModal;
  Release;
 end;
end;
```

memória pode ser desastroso se existirem variáveis do tipo TAbstractFactory ou qualquer uma de suas descendentes apontando para ela. Isto pode ser resolvido colocando esse método como private e garantindo que somente a finalization da unit possa invocá-lo.

A **Listagem 11** mostra a nova unit do form principal, agora reformulada. Veja como é diminuída a quantidade de código e as dependências de units, atingindo assim um dos principais objetivos propostos pelos padrões apresentados nesta série.

O interessante até aqui: a unit principal só precisa ter uFactorySingleton no seu uses. E podemos criar um formulário de cadastro já ligado com um objeto apenas chamando TFactorySingleton.Factory.CreateInterface para criar o Form e realizar a chamada com TFactorySingleton.Factory.CreateProduct para criar o produto já ligando com o form.

Instanciando classes pelo nome

Conforme foi citado no começo do artigo, será aplicada uma maneira de reduzir acoplamento. Basicamente será alterada a TFactorySingleton para que a mesma instancie um form através de seu nome. Para isso um objeto de cada uma das fábricas será armazenado em um TStringList. O método Find (ou indexOf) de TStringList retorna o objeto instanciado correspondente, contendo um Type Cast para a Factory desejada. Assim é possível criar Forms e outros produtos concretos baseados apenas no nome. Isto pode ser visto na **Listagem 12**.

Listagem 12. Controlando várias fábricas com TStringList

```
unit uFactorySingleton;

interface

uses
 uAbstractFactory,
 uPessoaFisicaFactory,
 uPessoaJuridicaFactory,
 classes;

type
 TFactorySingleton = class
 private
  class function CreateFactory: TAbstractFactory;
  //função que irá criar a factory efetivamente
  class function  GetFactory: TAbstractFactory; static;
  class var FLista: TStringList;
  class var FFactory: TAbstractFactory;
 public
  class function RetornaFactory(
   NomeClasse: string): TAbstractFactory;
  class procedure InicializaFactory;
  class procedure LiberarFactory;
  class property Factory: TAbstractFactory
  read GetFactory;
 end;

implementation

uses
 inifiles,
 forms;

{ TFactorySingleton }

class function TFactorySingleton.CreateFactory: TAbstractFactory;
begin
 with TIniFile.Create(Application.ExeName+'.ini') do
 try
  Result := RetornaFactory(
ReadString('Aplicacao', 'TipoEmpresa', ''));
 finally
  free;
 end;
end;

class function TFactorySingleton.GetFactory: TAbstractFactory;
begin
 //esse padrão é bom porque, embora a propriedade
 //esteja sempre acessível, ela será criada
 //somente uma vez e sob demanda
 if FFactory = nil then
 begin
  FFactory := CreateFactory;
 end;
 Result := FFactory;
end;

class procedure TFactorySingleton.InicializaFactory;
begin
 FLista := TStringList.Create;
 FLista.Sorted := True;
 FLista.AddObject('Atacado', TPessoaJuridicaFactory.Create);
 FLista.AddObject('Varejo', TpessoaFisicaFactory.Create);
 //novo tipo a ser implementado mais tarde
 //pode ser provisoriamente TPessoaFisicaFactory
 FLista.AddObject('NovoTipo', TPessoaFisicaFactory.Create);
end;

class procedure TFactorySingleton.LiberarFactory;
var i: integer;
begin
 for i := 0 to FLista.Count-1 do
 begin
  FLista.Objects[i].Free;
 end;
 FLista.Free;
end;

class function TFactorySingleton.RetornaFactory(
```

O que essa nova abordagem faz de interessante é que ela "pré-cria" uma fábrica concreta de cada tipo na inicialização, associando-as com nomes, criando assim uma tabela de chave-valor. Porém, cada nome pode aparecer uma única vez, e associado a cada um deles pode haver somente uma instância de uma fábrica concreta. Assim criamos instâncias de todas as fábricas concretas ao mesmo tempo, uma de cada tipo, sendo que todas continuam sendo Singletons.

Com isso aumentamos a flexibilidade do sistema e agora o arquivo ini pode ser modificado em runtime, com o programa aberto, que as alterações feitas nele refletirão imediatamente.

Com essas alterações pode-se tanto criar objetos das classes padrão, como também criar as classes através de seu nome, ou de um nome qualquer. Além disso, podemos colocar uma classe provisória no lugar de uma qualquer que ainda não terminamos de implementar. Essa classe provisória pode simplesmente ser um esqueleto onde todos os métodos disparam uma exception como também pode ser um wrapper para outra classe.

A **Figura 1** mostra o formulário principal usando agora um menu, sua codificação está na **Listagem 13**. Não é necessário praticamente referência nenhuma a nenhuma unit ou classe. Tudo o que precisamos são o TFactorySingleton e alguns nomes, em formato string.

Um dos itens do menu cria os objetos padrão conforme estiver no INI e os outros três têm seu evento OnClick tratado pelo mesmo método. O Item "Padrão do Sistema (INI)" usa a abordagem baseada no arquivo de configuração. Os outros três usam o método RetornaFactory e o Caption do menu para retornar a fábrica que tem exatamente o mesmo nome que o menu na lista de fábricas criadas. Esse padrão de projeto, além de auxiliar no desacoplamento, ajuda a transferir a complexidade ao longo das classes, dividindo cada vez mais a tarefa em tarefas menores.

Outra coisa importante é o item do menu chamado "NovoTipo", que não possui uma fábrica com esse nome, muito menos uma nova família de objetos. No entanto, a linha de código "FLista.AddObject('NovoTipo', TPessoaFisicaFactory.Create);" definiu que NovoTipo é somente um alias para TPesso-

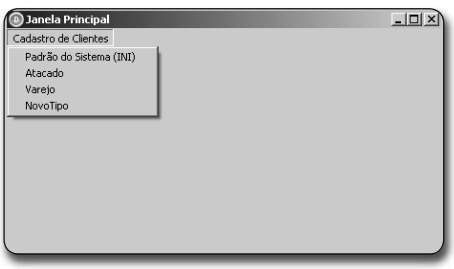


Figura 1. Os itens do menu em uso

Nota

Uma prática interessante seria criar um arquivo ou banco de dados de configuração que diz quais módulos do seu sistema o seu cliente pode acessar. Ao ler essa configuração o sistema cria em runtime os menus correspondentes, e cada menu executa o mesmo evento padrão, que é usar o Abstract Factory para criar o Form que está registrado com o mesmo nome que o Caption do item de menu.

Continuação: Listagem 12. Controlando várias fábricas com TStringList

```
  NomeClasse: string): TAbstractFactory;
var
  idx: integer;
begin
  if FLista.Find(NomeClasse, idx) then
    Result := (FLista.Objects[idx] as TAbstractFactory)
  else
    Result := nil;
end;

initialization
  //inicializar a lista
  TFactorySingleton.InicializaFactory;

finalization
  //liberar da memória a única instância
  //do objeto Concrete Factory (singleton)
  TFactorySingleton.LiberarFactory;
end.
```

Listagem 13. Formulário principal usa TFactorySingleton e nomes das classes como strings

```
unit Unit1;
...
  uFactorySingleton,
...
type
  TForm1 = class(TForm)
...
    procedure PadrodosisemaINI1Click(Sender: TObject);
    procedure NovoTipo1Click(Sender: TObject);
...
procedure TForm1.NovoTipo1Click(Sender: TObject);
begin
  with TFactorySingleton.RetornaFactory(
     (sender as TMenuItem).Caption).CreateInterface do
  begin
    CarregaCliente(TFactorySingleton.RetornaFactory(
(sender as TMenuItem).Caption).CreateProduct);
    ShowModal;
    Release;
  end;
end;

procedure TForm1.PadrodosisemaINI1Click(Sender: TObject);
begin
  with TFactorySingleton.Factory.CreateInterface do
  begin
    CarregaCliente(TFactorySingleton.Factory.CreateProduct);
    ShowModal;
    Release;
  end;
end;
```

Listagem 14. Nova Abstract Factory

```
unit uAbstractFactory;

interface

uses
 uFrmCadastroAbstrato;

type
 TAbstractFactory = class
 public
  function CreateInterface: TFrmCadastroAbstrato; virtual; abstract;
 end;

implementation

end.
```

Listagem 15. Nova TFactorySingleton

```
unit uFactorySingleton;

interface

uses
 uAbstractFactory,
 classes;

type
 TFactorySingleton = class
 private
  class var FLista: TStringList;
 public
  class function RetornaFactory(
     NomeClasse: string): TAbstractFactory;
  class procedure InicializaFactory;
  class procedure LiberarFactory;
 end;

implementation

uses
 forms,
 uPessoaFisicaFactory,
 uPessoaJuridicaFactory,
 uProdutosFactory,
 uVendasFactory;

class procedure TFactorySingleton.InicializaFactory;
begin
                              ...
 FLista.AddObject('Produtos', TProdutosFactory.Create);
 FLista.AddObject('Vendas', TVendasFactory.Create);
end;
...
```

Listagem 16. Form de cadastro básico

```
unit uFrmCadastroAbstrato;
...
type
 TFrmCadastroAbstrato =  class(TForm)
 end;

implementation

{$R *.dfm}

end.
```

aFisicaFactory. Isso mostra como você pode colocar alias para fábricas, encapsular fábricas com wrappers ou até usar uma fábrica de improviso enquanto outra não fica pronta.

Experiências com forms

Existem algumas particularidades interessantes do Delphi a demonstrar. Pode-se utilizar somente o Form de cadastro abstrato e seus descendentes concretos. É possível usar até mesmo a classe básica TForm do Delphi como Abstract Product.

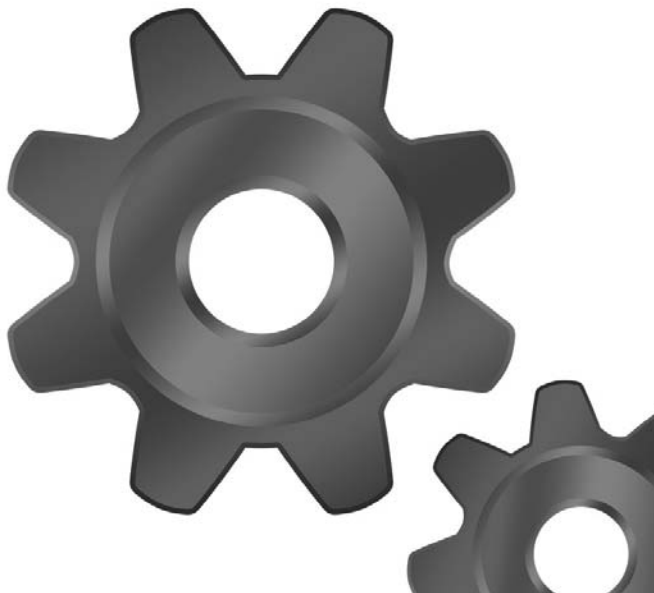
No exemplo podemos retirar do código tudo o que está relacionado aos produtos concretos e abstratos (clientes), a uAbstractProduct, a uTPessoaFisica e uTPessoaJuridica, com as devidas alterações necessárias para as outras units funcionarem sem elas. As fábricas concretas também não vão mais criar produtos que não sejam os Forms.

Considere agora vários forms adicionados, porém não criados na inicialização da aplicação. Todos são instanciados em runtime pela Abstract Factory. Todos são descendentes de TFrmCadastroAbstrato com um Concrete Factory para cada um.

A **Listagem 14** reflete como fica a nova Abstract Factory. TFactorySingleton não usa mais o INI, pois o que queríamos demonstrar já foi bem trabalhado. Não existe a propriedade Factory, o Field FFactory e o "acessor" GetFactory. O método CreateFactory também não é mais necessário.

Tudo o que é relacionado com os antigos produtos concretos TPessoaFisica e TPesoaJuridica também pode ser retirado, deixando apenas os formulários como objetos concretos. A **Listagem 15** mostra as modificações da classe TFactorySingleton e a **Listagem 16** apresenta o cadastro básico. O formulário de cadastro de pessoa física e o de pessoa jurídica simplesmente herdam de TFrmCadastroAbstrato, não tendo nenhum código adicional, nem mesmo os objetos concretos.

A **Figura 2** apresenta o formulário de cadastro de produtos descendente do form básico. A **Figura 3** mostra o form que simula vendas. Veremos como fábricas abstratas podem ser usadas na hora de se invocar formulários para consulta ou cadastro. Simularemos uma venda sendo feita, na hora de escolher ou cadastrar um cliente, a janela apropriada será aberta.



O formulário de vendas tem seu código exibido na **Listagem 17.** Note que este apenas chama os formulários de clientes e produtos para consulta ou cadastro, como se estivesse buscando ou inserindo clientes e produtos para uma venda. As fábricas de pessoas físicas e jurídicas agora criam apenas os formulários (**Listagens 18** e **19**). As fábricas de produtos da **Listagem 20** e de vendas da **Listagem 21** são novas, mas seguem a mesma linha de raciocínio que as demais, lembrando que nossas fábricas são todas fábricas de GUI, ou seja, formulários.

Usar Type Cast ou não dependerá de como estiverem organizadas suas classes e interfaces, mas o ideal é reduzir a zero o Type Cast por usar interfaces. Assim as units onde as classes estão implementadas podem ser trocadas ou alteradas sem que estejam no uses de outras units causando dependências.

Veja o que pode ser feito agora: chamar forms pelo nome, usando arquivos de configuração, xml, banco de dados. Se mesclar esse conhecimento com RTTI poderá, além de instanciar classes pelo seu nome, executar métodos ou disparar eventos usando apenas Strings.

Você deve estar se perguntando: "Eu tenho que criar uma Factory para cada Form ou classe minha? Isso é muito chato e dá muito trabalho!". Baseado no que já vimos, faremos mais um exemplo e implementação de Factory Method e Abstract Factory utilizando Generics.

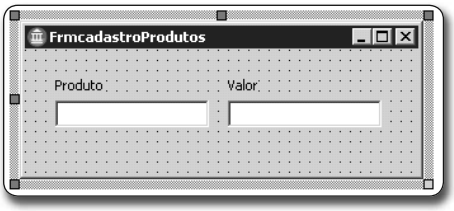


Figura 2. Cadastro de produtos

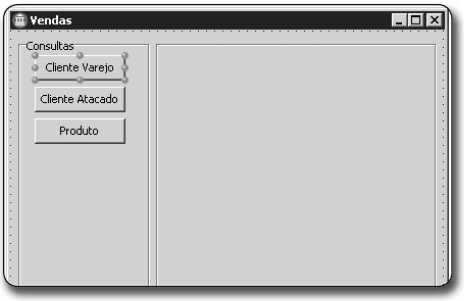


Figura 3. Formulário de vendas

Listagem 17. Novo form de vendas

```
unit uFrmVendas;
..
 TFrmVendas = class(TFrmCadastroAbstrato)
...
procedure TFrmVendas.Button1Click(Sender: TObject);
begin
 with TFactorySingleton.RetornaFactory('Varejo').CreateInterface do
 begin
  ShowModal;
  Release;
 end;
end;

procedure TFrmVendas.Button2Click(Sender: TObject);
begin
 with TFactorySingleton.RetornaFactory('Produtos').CreateInterface do
 begin
  ShowModal;
  Release;
 end;
end;

procedure TFrmVendas.Button3Click(Sender: TObject);
begin
 with TFactorySingleton.RetornaFactory('Atacado').CreateInterface do
 begin
  ShowModal;
  Release;
 end;
end;
```

Listagem 18. Fábrica de pessoa física

```
unit uPessoaFisicaFactory;
...
 TPessoaFisicaFactory = class(TAbstractFactory)
 public
  function CreateInterface: TFrmCadastroAbstrato; override;
 end;
...
function TPessoaFisicaFactory.CreateInterface: TFrmCadastroAbstrato;
begin
 Result := TFrmCadastroPF.Create(nil);
end;
```

Listagem 19. Fábrica de pessoa jurídica

```
unit uPessoaJuridicaFactory;
...
 TPessoaJuridicaFactory = class(TAbstractFactory)
 public
  function CreateInterface: TFrmCadastroAbstrato; override;
 end;
...
function TPessoaJuridicaFactory.CreateInterface: TFrmCadastroAbstrato;
begin
 Result := TFrmCadastroPJ.Create(nil);
end;
```

Listagem 20. Fábrica de produtos

```
unit uProdutosFactory;
...
type
 TProdutosFactory = class(TAbstractFactory)
 public
  function CreateInterface: TFrmCadastroAbstrato; override;
 end;
...
function TProdutosFactory.CreateInterface: TFrmCadastroAbstrato;
begin
 Result := TFrmcadastroProdutos.Create(nil);
end;
```

Listagem 21. Fábrica de vendas

```
unit uVendasFactory;
...
 TVendasFactory = class(TAbstractFactory)
 public
  function CreateInterface: TFrmCadastroAbstrato; override;
 end;
...
function TVendasFactory.CreateInterface: TFrmCadastroAbstrato;
begin
 Result := TFrmVendas.Create(nil);
end;
```

Nota do DevMan

Generics
Um dos novos recursos do Delphi, desde a versão 2009, são Generics. No Delphi XE tivemos ainda mais aprimoramentos e bugs corrigidos. Eles são similares aos templates do C++, mas não totalmente idênticos, por isso não vamos compará-los. São mais parecidos com os Generics do C#, então quem já está familiarizado com o ambiente .NET não sentirá dificuldades em usá-los no Delphi, e ainda ficará feliz em saber que agora podemos desfrutar desse recurso em um compilador de executáveis Win32 nativos.

Generics é um tipo de dado flexível. São tipos parametrizados, ou, mais claramente, parâmetros-tipo, que podem ser definidos em design. Não devemos confundir os parâmetros genéricos com os tipos propriamente ditos. Generics não são tipos, são parâmetros. Assim como os parâmetros ou argumentos de um método são substituídos por um valor, o parâmetro do tipo genérico é substituído por um tipo. Os Generics fornecem uma nova forma de flexibilização dos nossos sistemas, para somar-se às formas que nós já conhecemos, como interfaces, padrões de projeto e RTTI.

Podemos criar métodos com argumentos cujos tipos podem ser indefinidos, assim, podemos trocar o tipo de dado que o método usa cada vez que precisarmos, sem ter a obrigação de criar vários métodos diferentes. Também é possível criar classes que podem virtualmente se conectar com quaisquer outras, porque contém métodos ou propriedades que aceitam dados de qualquer tipo.

Você pode argumentar que sempre fez isso usando Variants quando o tipo de dado era um tipo primitivo, ou um objeto da classe TObject quando o tipo fosse uma classe qualquer. Porém, nesses casos, você era obrigado a usar os comandos da RTTI "is" e "as" para tentar fazer um TypeCast seguro, ou disparar uma exceção caso o tipo não fosse o correto. Nem sempre você conseguiria o tipo correto e isso era causa de erros no seu código.

Outras vezes, para flexibilizar a passagem de parâmetros entre os objetos, porém limitando para que não fosse usado qualquer tipo, nós usaríamos interfaces. Fazendo com que vários objetos implementassem a mesma interface você poderia intercambiá-los à vontade, ainda com a vantagem de acesso a todos os métodos da interface, do AutoComplete em tempo de design, no IDE, e da notificação de algum erro na hora da compilação.

Então, para que Generics?
Generics permitem que o tipo de dado passado seja um tipo primitivo, como Integer ou Double, ou um objeto de uma classe, ou ainda um tipo estruturado como um Record. Além disso, caso você informe o tipo errado, receberá a mensagem de erro na hora da compilação, e não em tempo de execução, com o programa em produção, causando um Access Violation. E ainda tem a vantagem de ter acesso a todas as informações do objeto em tempo de Design, através do AutoComplete.

Na maioria dos casos usam-se os tipos genéricos para se compor listas de objetos, ou objetos que contêm outros. O Delphi já vem com algumas classes em suas bibliotecas que usam Generics.

Listagem 22. TFrmCadastroAbstrato

```
unit uFrmCadastroAbstrato;
...
 TFrmCadastroAbstrato = class(TForm)
  public
   constructor Create; reintroduce; virtual;
  end;
...
constructor TFrmCadastroAbstrato.Create;
begin
  inherited Create(nil);
end;
```

Listagem 23. Fábrica concreta com Generics

```
unit uConcreteFactory;

interface

uses
  uFrmCadastroAbstrato,
  uAbstractFactory;

type
  TConcreteFactory<T:TFrmCadastroAbstrato,
constructor> = class(TAbstractFactory)
  public
    function CreateInterface: TFrmCadastroAbstrato;
     override;
  end;

implementation

{ TConcreteFactory<T> }

function TConcreteFactory<T>.CreateInterface:
  TFrmCadastroAbstrato;
begin
  Result := T.Create;
end;
```

Nota

Se você passar Application como owner no constructor de um Form você pode garantir que não haverá Memory Leaks, mas se você esquecer o Release, ou Free, os Forms criados com Application como Owner só serão realmente liberados da memória após o fechamento da aplicação. Isso faria com que o programa utilizasse cada vez mais memória e no final, quando fosse fechado, teria uma grande quantidade de memória para liberar. Ele conseguiria liberar tudo, sem causar Leaks, mas apenas no fechamento. Isso pode ser um tipo invisível e perigoso de Leak se sua aplicação não for um simples programa desktop, mas sim um server de DataSnap 24 x 7 ou algum monitor de equipamentos industriais. Além disso, mesmo que você não se esqueça do Release, passando Application como Owner há o overhead de a cada criação do form incluir a mesma na coleção de componentes de Application. Tudo depende do seu projeto.

Mesclando Factory Method com Generics

Todas as Concrete Factorys do exemplo não são mais necessárias, usaremos apenas a unit da Abstract Factory. Para que um tipo genérico possa criar um objeto da classe que está sendo passada como parâmetro genérico é necessário que se use a constraint "constructor". Para utilizar a constraint constructor é obrigatório que a classe tenha um constructor padrão sem parâmetros. Os Forms e derivados possuem esse construtor, mas ele não inicializa certos atributos internos da classe, o que impede que o form seja exibido com ShowModal.

TFrmCadastroAbstrato (**Listagem 22**) tem agora um construtor especial, para que possa ser construído sem parâmetros. Para isso deve ser criado um construtor "falso" para os formulários que não tenham parâmetros, mas internamente será chamado o construtor ancestral passando nil como owner, com isso é necessário destruir o formulário em seguida. Isso pode ser feito também com a chamada Release no evento OnClose. O construtor é sobrecarregado com override, que é virtual, e o reintroduce garante que compilador aceite sobrecarregar um construtor com parâmetros por um sem parâmetros. A **Listagem 23** mostra a única fábrica concreta descendente de TAbstractFactory e que usará Generics.

A linha TConcreteFactory<T:TFrmCadastroAbstrato, constructor> ... declara uma classe que usará um tipo genérico T onde são passadas duas constraints: T precisa ser do tipo TFrmCadastroAbstrato ou de um de seus descendentes e T precisa ser um constructor. Se não for passada a constraint constructor não é possível, dentro do Factory Method CreateInterface, criar um objeto do tipo T com T.Create.

A Factory Singleton, no InicializaFactory, pode agora usar a fábrica concreta genérica, com FLista.AddObject('nome', TConcreteFactory<classe de um formulario>.Create), como mostra a **Listagem 24**.

Cada uma das cinco instâncias de TConcreteFactory armazenadas nos objetos da TStringList foram instanciadas usando um form diferente do projeto no lugar do parâmetro genérico T. O método CreateInterface de cada uma dessas instâncias criará um objeto da classe correta, que foi passada em T. Com isso, tem-se uma maneira mais fácil de implementar a Abstract Factory. Repare que poucas units foram alteradas.

Listagem 24. TFactorySingleton usando Generics

```
...
class procedure TFactorySingleton.InicializaFactory;
begin
 FLista := TStringList.Create;
 FLista.Sorted := True;
 FLista.AddObject('Atacado',
   TConcreteFactory<TFrmCadastroPJ>.Create);
 FLista.AddObject('Varejo',
   TConcreteFactory<TFrmCadastroPF>.Create);
 FLista.AddObject('NovoTipo',
   TConcreteFactory<TFrmCadastroPF>.Create);
 FLista.AddObject('Produtos',
   TConcreteFactory<TFrmCadastroProdutos>.Create);
 FLista.AddObject('Vendas',
   TConcreteFactory<TFrmVendas>.Create);
end;
```

Conclusão

Com flexibilidade podemos criar sistemas facilmente adaptáveis a uma gama maior de negócios e a requisitos mais flexíveis. Os padrões de projeto são ferramentas essenciais para conhecermos e utilizarmos no dia-a-dia adaptando-os às nossas necessidades e agregando valor aos projetos pelo uso de modelos bem consolidados e testados no mercado, sem "reinventar a roda".

É possível criar o objeto da classe que quiser, partindo unicamente de uma String ou entrada do usuário. Esta é a técnica máxima de abstração, que permite que o usuário crie ou modifique formulários, por exemplo, e é a mesma técnica usada no IDE do próprio Delphi ou do Rave.

Mas flexibilizar as interfaces com o usuário e as classes de negócio mantendo rígidas nossas classes de persistência é um contrassenso. ERPs modernos podem ser adaptados com os principais bancos de dados do mercado. Por isso, no próximo artigo, veremos não só como fazer um sistema multi-banco, mas "multi-tecnologia-de-acesso".

Esperamos que com esse artigo o leitor esteja apto a criar formulários com baixo acoplamento entre si, pouco dependentes uns dos outros. Também é possível derivar desse artigo um mecanismo para centralizar a chamada e abertura de formulários, ou a criação de qualquer classe e até componentes.

Vitor Luiz Rubio
vitorrubio@gmail.com
www.vitorrubio.blogspot.com
Tecnólogo em Processamento de Dados pela FATEC-SP e MCP. Atua como Analista de Sistemas Sênior na Editora Revista dos Tribunais. Programa com Delphi desde a versão 3 e com .Net desde a versão 1.1.

Blogs do autor
http://www.vitorrubio.blogspot.com
http://twitter.com/vitorrubio

Dê seu feedback sobre esta edição!

A Clubedelphi tem que ser feita ao seu gosto. Para isso, precisamos saber o que você, leitor, acha da revista!

Dê seu voto sobre este artigo, através do link:
www.devmedia.com.br/clubedelphi/feedback

Faça um *upgrade* em sua carreira

Em um mercado cada vez mais focado em qualidade, ter conhecimentos aprofundados sobre requisitos, metodologia, análises, testes, entre ouutros, pode ser a diferença entre conquistar ou não uma boa posição profissional. Sabendo disso a DevMedia lançou uma publicação totalmente especializada em Engenharia de Software. Todos os meses você pode encontrar artigos sobre Metodologias Ágeis; Metodologias tradicionais (document driven); ALM (application lifecycle); SOA (aplicações orientadas a serviços); Análise de sistemas; Modelagem; Métricas; Orientação à Objetos; UML; testes e muito mais. **Assine Já!**

DevMedia

Delphi Prism e AS

Reaproveite seus conhecimentos Win32 / De

Quem desenvolve em Delphi Win32 e precisou implementar o controle de usuários deve lembrar que a VCL não possui nenhum framework pronto para isso. No ASP.NET 1.x (Visual Studio 2003), também não existiam controles para implementar seguranças de Web Sites. Apesar de haver uma API, era necessário implementar muita coisa no braço, principalmente telas.

O ASP.NET 2.0 foi a versão do ASP.NET que mais trouxe recursos de todas as versões do framework, juntamente com o Visual Studio 2005. Uma dessas grandes novidades foi o incrível suporte facilitado à implementação de segurança de um Web Site. Por exemplo, imagine as tarefas mais básicas que precisam ser realizadas por qualquer site na internet: validar usuários, registrar novos, permitir troca de senha, oferecer uma forma de recuperação de senha, exibir determinadas partes do site ou não de acordo com as credenciais do usuário etc. Tudo isso precisava ser feito via código no ASP.NET 1.x, apesar do framework ter classes prontas para isso. A diferença é que no ASP.NET 2.0 foram inclusos novos controles servidores em uma guia chamada Login que permitem realizar todas as tarefas citadas anteriormente, sem uma linha de código sequer. Podemos citar aí os controles Login, LoginName, LoginStatus, LoginView, ChangePassword, CreateUserWizard e PasswordRecovery, alguns dos quais conheceremos neste artigo.

Em nosso site, o usuário precisa em certo ponto criar uma conta para poder finalizar a compra, e aqui vai uma dica, jamais obrigue o usuário a criar um cadastro na primeira visita, deixe ele navegar pelo site, pesquisar produtos, adicionar ao carrinho, e somente em momento oportuno permita a criação de um cadastro. É exatamente isso que faremos nesta etapa. Além disso, precisamos definir uma área protegida para o site. Ou seja, após o usuário logar (processo chamado de autenticação), precisamos ver quem pode acessar determinadas páginas (processo chamado autorização).

Resumo DevMan

De que trata o artigo:

Lançado pela Embarcadero, o Delphi Prism é a nova forma de desenvolver aplicações para o .NET Framework e Mono utilizando o Object Pascal e o IDE do Visual Studio / MonoDevelop. Nesta série de artigos, veremos como utilizar o Delphi Prism para criar um site de E-Commerce usando a tecnologia ASP.NET.

Para que serve:

O objetivo principal desta série de artigos é familiarizar os desenvolvedores Delphi Win32 com o ambiente de desenvolvimento do Delphi Prism, o IDE do Visual Studio, bem como introduzir o desenvolvimento de aplicações Web. Dessa forma, faremos vários comparativos entre os dois IDEs e também entre os Frameworks ASP.NET e VCL, servindo como um excelente guia para diminuir a curva de aprendizado ao se aprender ASP.NET baseando-se nos seus prévios conhecimentos em Delphi Win32 (Desktop).

Em que situação o tema é útil:

Esta série será muito útil para quem nunca desenvolveu aplicativo algum para Web, nem mesmo teve um primeiro contato com o Delphi Prism / Visual Studio. Ao desenvolver aplicações para Web, ganhamos inúmeras vantagens se compararmos ao ambiente Desktop: Deploy simplificado, uso de computação nas nuvens, não há necessidade de instalações visto que os usuários acessam um mesmo servidor na internet, não há dependência do sistema operacional, pois usamos um browser, não há preocupação com backups nos clientes, nem altos custos para manutenções e visitas aos clientes etc.

Delphi Prism e ASP.NET:

No primeiro artigo desta série, tivemos uma introdução ao Delphi Prism, seus objetivos e propósitos. A seguir partimos para a criação do banco de dados no SQL Server usado ao longo desta série. Na segunda parte iniciamos a criação do Web Site, aproveitando para falar dos fundamentos do ASP.NET e Visual Studio, sempre comparando-os com a VCL e RAD Studio, com o intuito de facilitar o aprendizado. Vimos a estrutura básica de um Web Site e iniciamos a construção do layout básico do site de E-Commerce (chamado "Computer Store"), bem como o sistema de navegação. Na parte 3, vimos como construir a página principal do site, exibindo a lista de produtos, mostrando como fazer a conexão com o banco de dados. Na parte 4 vimos dois importantes passos do site, tarefas comuns realizadas em qualquer tipo de aplicação Web que trabalhe com banco de dados, como pesquisas com parâmetros. Na parte anterior, vimos como adicionar uma das principais funcionalidades do site, a possibilidade do usuário adicionar produtos a um carrinho de compras, onde usamos ADO.NET. Nesta 6ª parte da série, veremos um recurso muito interessante e que possui um excelente suporte pronto e nativo do ASP.NET: segurança com os controles de Login.

Criando a página de Login

No Solution Explorer, dê um clique de direita sobre o nome do Web Site, escolha a opção Add New Item>Web Form e dê ao novo formulário o nome de Login.aspx, desmarcando a opção para selecionar a MasterPage. Neste formulário coloque um

P.NET

sktop para criar aplicações Web – Parte 6

Guinther Pauli

controle Login da paleta de mesmo nome. Na opção Auto Format na Smart Tag do controle, escolha uma formatação ao seu gosto. Na propriedade DestinationPageUrl informe ~/CheckOut/Checkout.aspx (não criamos ainda), que é a página para onde o usuário será redirecionado após efetuar o Login.

Veja que temos várias opções para configurar o controle, que é totalmente personalizável. É possível editar os títulos dos textos, como nome do usuário e senha. Se você escolher a opção Convert to template na Smart Tag, o Login será ajustado de forma a permitir a alteração direta nos controles internos, como os Labels, TextBoxs e controles de validação.

Da mesma forma que fizemos para a página de Login, adicione mais uma página chamada NewUser.aspx. Nela coloque um controle CreateUserWizard com a formatação ao seu gosto. Na propriedade ContinueDestinationPageUrl informe o valor ~/Login.aspx, que é a página que o usuário será redirecionado após efetuar seu cadastro.

De volta à página de Login, configure as propriedades CreateUserText para New user e CreateUserUrl para ~/NewUser.aspx, o que vai configurar o controle para exibir um texto para a criação de uma nova conta e automaticamente redirecionar o usuário para a página de cadastro.

Testando o Login e gerando o banco de dados de usuários

No Solution Explorer dê um clique de direita sobre a página Login.aspx e escolha a opção Set as Start Page, para que ela seja a página inicial a ser exibida quando a aplicação for executada, o que faremos neste momento (Ctrl+F5). Informe dados quaisquer na tela de Login e tente acessar o site. Certamente uma mensagem de Login inválido aparecerá, como mostra a **Figura 1**. A **Figura 2** mostra a página para criar um novo usuário (mais adiante veremos como configurar o Login para acessá-la).

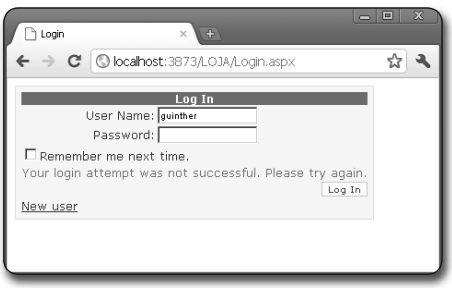


Figura 1. Testando o Login

Figura 2. Criando um novo usuário

Uma certa demora ocorrerá na primeira vez que o usuário tentar logar. Isso não acontece por acaso. O ASP.NET neste momento criou automaticamente um banco de dados local, chamado ASPNETDB.MDF, na pasta App_Data, para armazenar os dados sobre os usuário. Você pode explorar a estrutura desse banco de dados através do Solution Explorer / Server Explorer.

Web Site Administration Tool

Acesse então o menu Web Site > ASP.NET Configuration, opção que também está disponível no controle Login em sua Smart Tag, e também como um botão no Solution Explorer. Será aberta a página de administração do Web Site (**Figura 3**), onde podemos configurar várias opções do Web Site, inclusive a segurança (Security). Clique no link desta opção, a seguir acesse o link Use the security Setup Wizard to configure security step by step, vá navegando pelo assistente até chegar na opção para escolher o tipo de autenticação, onde devemos escolher From the internet. Isso mais precisamente vai configurar a tag Authentication para Forms, no Web.Config:

```
<authentication mode="Forms" />
```

Nota do DevMan

O controle CreateUserWizard é bastante poderoso. Ele permite, por exemplo, que você omita a opção para que o usuário informe uma senha durante o cadastro. Nesse caso, o ASP.NET pode enviar um e-mail para o usuário após o registro, com uma senha gerada temporariamente, de forma que é necessária uma alteração de senha após o primeiro Login. Isso garante, entre outras coisas, que o e-mail digitado é válido e existe, evitando a gravação de dados incorretos ou inconsistentes no banco de dados. Essa funcionalidade poder ser obtida através da propriedade AutoGeneratePassword.

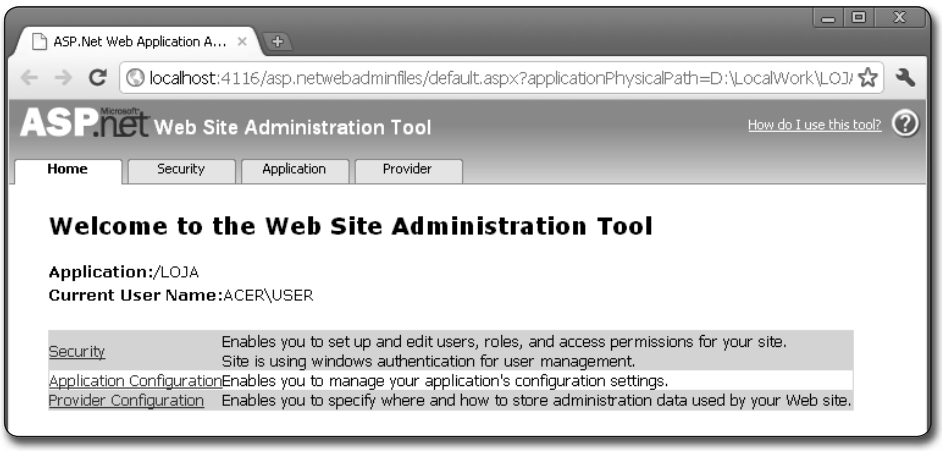


Figura 3. Web Site Administration Tool

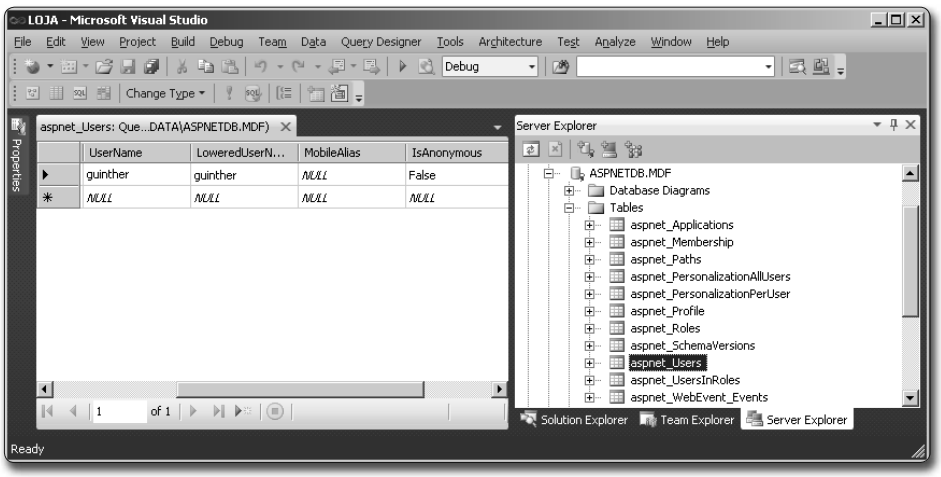


Figura 4. O banco ASPNETDB

Nota do DevMan

O assistente que você vê na configuração de segurança do Web Site Administration Tool é criado através do uso do controle Wizard do ASP.NET. Você pode usar este controle, que está disponível na guia Standard, para criar seus próprios assistentes em suas páginas. Este controle também foi introduzido originalmente no ASP.NET 2.0 / Visual Studio 2005.

Listagem 1. Negando acesso anônimo à pasta

```
<?xml version="1.0" encoding="utf-8"?>
<configuration>
  <system.web>
    <authorization>
      <deny users="?" />
    </authorization>
  </system.web>
</configuration>
```

Dessa forma, dizemos ao ASP.NET que os usuários do site serão autenticados através de nosso formulário personalizado, e não mais usando a autenticação integrada do Windows, onde o usuário que logou toma a identidade utilizada (padrão), que funciona somente em intranets.

Na tela de criação de usuários, adicione uma nova conta para testes, informando os dados. Note que a senha segue uma constraint do ASP.NET, deve ter 8 dígitos incluindo números, letras e símbolos. Este controle que você vê na ferramenta é exatamente o mesmo CreateUserWizard que adicionamos anteriormente em nosso Web Site. Neste momento você já pode finalizar o assistente. Os dados cadastrados serão armazenados automaticamente no banco ASPNETDB.

Existem outras configurações que você pode acessar através deste utilitário, muitas das quais afetarão diretamente o Web.Config. É possível, por exemplo, gerenciar usuários, configurar opções de servidor de E-Mail (SMTP), opções de Debug e Trace, personalização de Providers etc.

Neste ponto você já pode executar novamente a aplicação e testar o Login usando a conta criada. Acesse também o site e crie um novo usuário usando nossa página NewUser, clicando nesta opção no controle de Login. Veja que os dados serão cadastrados no banco ASPNETDB (veja a estrutura do banco na **Figura 4,** onde também é possível ver o usuário criado).

Configurando uma pasta para acesso restrito

Setar a página Login.aspx como "Start Page" não funciona como garantia de que o usuário vá sempre logar antes de fazer o check-out (finalizar a compra). Ele pode perfeitamente abrir o browser e digitar a URL da página que quiser, burlando o Login. A opção Set as Start Page apenas indica ao IDE do Visual Studio qual página executar primeiro quando pressionamos F5 ou Ctrl+F5 (ou menu Start). Precisamos criar uma área do site que seja protegida, acessível somente por usuários autenticados. Isso pode ser feito através do uso do recurso chamado Access Rules, que tem relação com a autorização.

Clique então com o botão direito no Solution Explorer e escolha New Folder, dando à pasta o nome de Checkout. Dê um clique de direita nesta pasta e adicione uma página chamada Checkout.aspx, herdando da MasterPage. Da mesma forma, crie uma página chamada Finish.aspx.

Novamente dê um clique com o botão direito no Solution Explorer e escolha a opção Add New Item>Web Configuration File. Sim, é possível ter mais de um Web.Config por Web Site, desde que estejam em pastas diferentes. Neste caso, o ASP.NET "herda" todos os elementos declarados na configuração da pasta raiz. A **Listagem 1** mostra o código que você deve digitar no arquivo. Aqui simplesmente definimos a tag Authorization, onde informamos que o acesso a usuários anônimos, ou seja, não autenticados (símbolo "?"), não é permitido.

Agora faça um teste, execute a aplicação e tente acessar diretamente a página, ou clique de direita sobre ela no Visual Studio, escolhendo View in Browser para acesso direto. Você será automaticamente redirecionado para a página de Login! E mais, observe o formato da URL gerada neste momento:
http://localhost:3873/LOJA/login.aspx?ReturnUrl=/LOJA/Checkout/Checkout.aspx

Notou o parâmetro ReturnUrl na Query String? É exatamente o que você está imaginando, o ASP.NET sabe que tentamos um acesso à página de Checkout, então nada melhor do que depois de ter exigido o Login do usuário, devolver a ele a página originalmente solicitada, processo também feito automaticamente.

FAQs

Você viu que o ASP.NET faz tudo praticamente sozinho, como se fosse mágica. Quando trabalhamos com o ASPNETDB, ou melhor, com os Application Services do ASP.NET, algumas dúvidas comuns surgem. Abaixo enumerei as principais:

É possível utilizar outro banco de dados que não o ASPNETDB?

Sim, poderíamos ter criado a estrutura desse banco em nosso BD previamente criado para armazenar os dados dos produtos e compras. Para isso, é necessário registrar os Application Services do ASP.NET no banco desejado, através da opção de linha de comando aspnet_regsql.

É possível utilizar os Application Services, incluindo controles de Login, com outro tipo de servidor de banco de dados?

Sim, mas para isso é necessário implementar um provider personalizado para MemberShip. Nesse caso, é possível utilizar todos os controles e serviços do ASP.NET com o Firebird, por exemplo, ou ainda utilizar uma estrutura de banco no SQL Server diferente da apresentada pelo ASPNETDB, ou ainda utilizar um banco de dados legado com informações de usuários.

Conclusão

Isso finaliza esta etapa da nossa série. Como pudemos comprovar, o ASP.NET dá uma mão na roda quando precisamos implementar a segurança de um Web Site, mesmo para realizar as tarefas mais exigentes e que antes necessitavam de codificação extra. Criar usuários, gerenciar contas, permitir troca de senha, recuperação, criação de banco de dados, tudo é feito automaticamente. Não é necessário digitar uma linha de código sequer. Na próxima etapa da série finalizaremos o processo de compra criando a página de Checkout. Grande abraço e até lá.

Nota do DevMan

O ASP.NET 2.0 introduziu o conceito de "Providers". Podemos dizer que um Provider nada mais é do que uma classe concreta que implementa uma classe abstrata, usado para comunicar duas partes de um sistema ASP.NET, de forma a tornar um serviço "plugável". Falando de forma mais prática, veja por exemplo os serviços relacionados à segurança (Membership). Quando usamos um controle de Login (o que já vem pronto de fábrica), fornecemos as credenciais de acesso do usuário e tentamos entrar no site, o ASP.NET precisa fazer a validação do mesmo. Por padrão, o ASP.NET usa uma API pronta que usa um banco de dados no SQL Server para realizar a maioria dessas tarefas, como o cadastro de usuários, preferências etc. Porém, e se quisermos mudar esse comportamento, usando um outro banco de dados ou mesmo outra tabela? Se não fossem os "Providers", você teria que fazer tudo no braço de novo e descartar todos os controles de Login. Porém, o ASP.NET não chama funcionalidades de uma classe concreta para realizar essas tarefas, ele se comunica com um "Provider", através de uma classe abstrata, o que significa que você pode implementar uma nova classe concreta que herde da mesma classe abstrata e implemente as funcionalidades de segurança de outra forma, por exemplo, usando um banco Firebird, reaproveitando os controles de Login. Criada a classe customizada, basta dizer ao ASP.NET, no Web.Config, qual o nome da classe. De fundo, o ASP.NET faz com isso uso de uma boa prática de desenvolvimento, implementada através de um Design Pattern chamado Abstract Factory.

Site oficial do Delphi Prism
http://www.embarcadero.com/products/delphi-prism

Wiki do Delphi Prism
http://prismwiki.codegear.com/en/Main_Page

Download do Delphi Prism
https://downloads.embarcadero.com/free/delphi_prism

Site oficial do ASP.NET
http://www.asp.net

Visual Studio
www.microsoft.com/visualstudio

Blogs do autor
http://twitter.com/guintherpauli
http://gpauli.com

Guinther Pauli
guinther@devmedia.com.br
guinther.pauli@gmail.com
http://twitter.com/guintherpauli
http://gpauli.com

Desenvolve em Delphi há 15 anos, tendo utilizado a ferramenta desde a versão 1, passando pelo Turbo Pascal e Borland Pascal for Windows. Certificado em Delphi 3, 5, 6, 7, 2005, 2006, Delphi para Web e Delphi para Linux, palestrante em 5 edições da Borland Conference. Certificado Microsoft MCP, MCAD, MCSD.NET, MCTS e MCPD (C#, ASP.NET), utilizando a plataforma .NET desde a versão beta. Editor das Revistas ClubeDelphi, .NET Magazine Brasil e EASY .NET Magazine, professor de Pós-Graduação da UNIVEM-SP, na área .NET. Especialista em desenvolvimento Delphi, C#, .NET, ASP.NET, Visual Studio, atuando também em projetos de migração, orientando empresas e desenvolvedores na adoção de melhores práticas, ministra treinamentos em .NET focados em equipes Delphi. Foi arquiteto e desenvolvedor ASP.NET do site de doações on-line do projeto Criança Esperança da Rede Globo / Unesco.

Dê seu feedback sobre esta edição!

A Clubedelphi tem que ser feita ao seu gosto. Para isso, precisamos saber o que você, leitor, acha da revista!

Dê seu voto sobre este artigo, através do link:

www.devmedia.com.br/clubedelphi/feedback

Entendendo e tratando

Desenvolva aplicações robustas, eficientes

Quem nunca recebeu a mensagem "Esta aplicação executou uma operação ilegal e será finalizada" que atire a primeira pedra. As exceções não tratadas e aplicações vulneráveis a elas não são privilégio de programadores amadores, pelo contrário, temos hoje no mercado aplicativos de nome que por vezes nos tiram do sério. Mas o que vem a ser estas exceções? Antes de responder esta pergunta temos primeiro que entender como funciona uma aplicação nos bastidores.

Em sua maioria, os aplicativos que rodam no Windows possuem a extensão .exe, são os famosos arquivos executáveis, o principal responsável por fazer acontecer. Basicamente para produzir um executável você precisa apenas de um compilador. O papel do compilador é traduzir algo escrito em uma linguagem digamos humana para a linguagem que o computador, ou melhor, o processador consiga entender: zero e um. Este processo é chamado de compilação.

Imagine que você queira fazer um pequeno aplicativo que fornecida uma idade informe se a pessoa é menor. Seria muito bom se pudéssemos passar a seguinte instrução para o processador: Processador se eu lhe passar um valor e este valor for menor do que 18 por gentileza envie o texto "É menor de idade" para o monitor. O processador não "fala" português, ele entende apenas zero e um, então precisamos de um compilador que possa converter, ou melhor, traduzir o texto acima para zero e um. Bem eu não conheço um que converta texto em português para o que chamamos de linguagem de máquina, mas temos o Delphi, que possui um compilador para a linguagem Delphi. Desta forma a mensagem acima ficaria mais ou menos assim:

Resumo DevMan

De que trata o artigo:

O artigo trata de um assunto por muitas vezes ignorado pela maioria dos desenvolvedores, o tratamento de exceção. Vamos entender o processo de compilação, pilha de processamento e por fim o que são as exceções e onde elas se enquadram no contexto da execução de um programa para que assim possamos implementar um tratamento e proteção de recursos eficiente.

Para que serve:

O tratamento de exceção serve para tornar as aplicações mais tolerantes a falhas, não significa dizer que sua aplicação estará isenta de erros, mas sim que ela estará preparada para lidar com possíveis erros, seja garantindo a integridade dos dados, gerando logs, notificando usuários ou tomando outra atitude que evite o travamento da aplicação.

Em que situação o tema é útil:

A implementação de uma rotina para tratar exceções é útil quando se quer garantir que procedimentos da sua aplicação sejam executados em sua totalidade, e quando por algum motivo isso não for possível que a pilha de processamento não seja interrompida ocasionando o travamento da aplicação.

Exceções no Delphi:

Neste artigo vamos entender o que são exceções e como elas são encaradas pelo Delphi. O objetivo principal é mostrar a você que com atitudes simples podemos reduzir consideravelmente aquelas mensagens incômodas de erros que tiram qualquer usuário do sério. Vamos desenvolver um exemplo mostrando algumas técnicas interessantes de tratamento de exceções, tais como rotinas para extrair informações dos erros e nos auxiliar na manutenção dos nossos sistemas permitindo-nos aprimorá-los, tornando-os cada vez mais tolerantes a falhas.


```
If Idade < 18 then
  ShowMessage('É menor de idade');
```

Isso é suficiente para que o compilador do Delphi entenda e traduza isso em linguagem de máquina para que o processador possa entender essa instrução. No Delphi isso é feito apenas pressionando as teclas Ctrl+F9, porém uma série de tarefas são executadas até que o produto final, o executável, seja criado. Feito isso temos nossa aplicação pronta para ser usada e ao darmos um duplo clique no executável podemos usufruir o benefício de saber, dada a idade de uma pessoa, se a mesma é menor de idade. Falando assim o processo de compilação parece simples, mas há fases pelas quais o código passa até chegarmos ao produto final. O processo de compilação é composto de análise e síntese. A análise tem como objetivo entender o código fonte e representá-lo em uma estrutura intermediária. A síntese constrói o código objeto a partir desta representação intermediária. A **Figura 1** ilustra de forma resumida o processo de compilação.

Você pode estar se perguntando, o que a compilação tem a ver com tratamento de exceções? Bem, pelo exposto anteriormente podemos notar que uma vez compilado o executável não pode ter as linhas de código que o gerou alteradas, assim temos que

as exceções no Delphi

e tolerantes a falhas

Rodrigo Carreiro Mourão

garantir que as instruções, ou melhor, que o código que escrevemos está apto a lidar com situações excepcionais, ou seja, com as exceções que por ventura venham ocorrer. Um exemplo clássico seria, no código citado anteriormente, alguém informar uma idade não numérica, uma letra no lugar de um número. Nossa instrução não trata isso, ou seja, informar uma letra quando se espera um número não é o normal é a exceção à regra e este fato, no executável já compilado e que não pode ser mudado, o torna falho mediante este fato. Agora pare e pense! Quantas instruções possuem as aplicações que criamos? Quantas são as instruções escritas e que estão sujeitas a estes tipos de exceções? E por fim quantas delas possuem o tratamento necessário?

Nota do DevMan

A análise pode ser subdividida ainda em análise léxica, análise sintática e análise semântica. A síntese é mais variada, podendo ser composta pelas etapas de geração de código intermediário, otimização de código e geração de código final (ou código de máquina), e somente esta última etapa é obrigatória. Classicamente, um compilador traduz um programa de uma linguagem textual facilmente entendida por um ser humano para uma linguagem de máquina, específica para um processador e sistema operacional. Atualmente, porém, são comuns compiladores que geram código para uma máquina virtual que é, depois, interpretada por um interpretador.

Nota

O objetivo deste artigo não é se aprofundar no processo de compilação, porém é importante ressaltar que, se tratando de Delphi o executável não é produto da compilação, e sim da linkedição. No Delphi a compilação gera os arquivos DCU (Delphi Compiled Unit), estes arquivos são linkeditados para que então seja gerado o produto final, em sua maioria um executável. O tipo de arquivo gerado em cada fase pode variar de acordo com o tipo de projeto, no caso de um pacote, por exemplo, o produto da compilação será um arquivo com extensão DCP (Delphi Compliled Pacakge) e o produto final da linkedicao de um pacote será um arquivo com extensão BLP (Borland Library Package).

Exceções no Delphi

Exceção em qualquer linguagem de programação é algo que altera o fluxo normal de um programa. Quando uma exceção ocorre o estado atual da aplicação é salvo em um local pré-definido na memória e o fluxo é direcionado para uma rotina de tratamento de exceção, quando esta existe, para que logo após o estado da aplicação possa ser restaurado e o processo continuar de onde parou. No Delphi as exceções nada mais são do que objetos do tipo Exception ou algum outro tipo descendente deste. Quando uma exceção ocorre um objeto é criado de acordo com o tipo de exceção gerado. Quando há um tratamento de exceção na rotina que gerou o erro o objeto da exceção é destruído e a pilha de execução continua de onde parou.

O problema está no fato de não haver um tratamento para uma eventual exceção. Quando isso ocorre, a pilha de execução da aplicação é interrompida, fazendo com que o código posterior à exceção não seja executado. Agora suponha que tenhamos uma rotina que lê as linhas de um arquivo

Nota do DevMan

Linkedição é o processo pelo qual um programa chamado linkeditor liga os objetos gerados por um compilador formando assim o arquivo executável final. O trabalho de um linkeditor é bastante simples, ele estabelece a amarração entre referências abstratas para referências concretas. Por exemplo, o programador chama o método ConverteValores, o linkeditor pega esta referência e faz o vínculo para o ponteiro que está 620 bytes após o início do módulo executável.

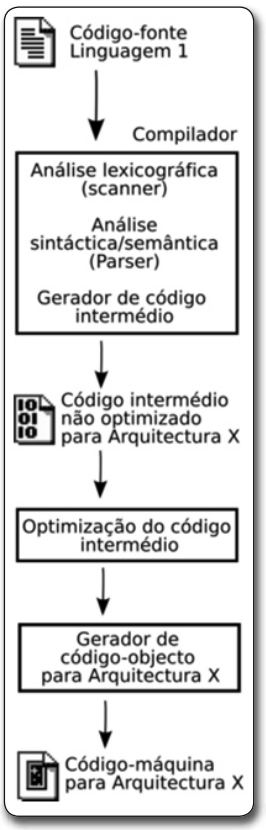
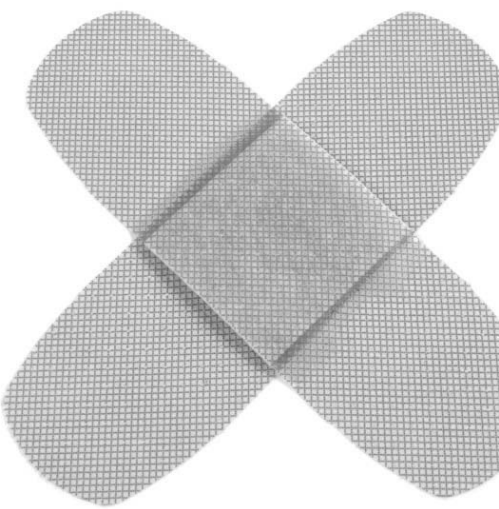


Figura 1. Processo de compilação



.txt qualquer e para cada linha deste arquivo execute um procedimento. Se na metade do arquivo ocorre um problema qualquer, todo o restante do arquivo ficará sem ser processado e aí temos um grande problema. Parte do arquivo foi processado e parte ficou de fora e o pior de tudo, não temos como descobrir a priori a linha exata onde o problema ocorreu. Neste caso dizemos que o código utilizado para fazer a leitura deste arquivo texto é um código não seguro. Um código seguro no contexto do tratamento de exceções é aquele que não produz efeitos prejudiciais, como vazamentos de memória, corrupção de dados ou saída inválida, mesmo quando ocorrem falhas em tempo de execução. Os níveis de segurança de um código podem ser classificados conforme exposto na **Tabela 1.**

Nível	Descrição
Transparência à falha	Há garantia que as operações ocorrerão com sucesso e satisfarão todos os requerimentos, mesmo na presença de situações excepcionais. Este é o melhor nível de segurança.
Transacional	As operações podem falhar, mas quando isso ocorre as operações não causam efeitos colaterais.
Segurança básica	Execuções parciais de operações que falham podem causar efeitos colaterais, mas os invariantes de estado são preservados, isto é, qualquer dado gravado conterá valores válidos.
Segurança mínima	Execuções parciais de operações que falham podem gravar dados inválidos, mas não levarão à falha completa (crash) do programa.
Sem segurança	Não há qualquer garantia. Este é o pior nível de segurança para com exceções.

Tabela 1. Níveis de segurança

Assim como uma exceção pode ocorrer ela também pode ser gerada. Lançar uma exceção é um modo útil de assinalar que a rotina não deve continuar a execução quando, por exemplo, os argumentos de entrada não são válidos ou quando um recurso no qual o programa depende não está disponível. No Delphi lançamos uma exceção através da palavra reservada raise. Já para tratarmos uma exceção no Delphi utilizamos o bloco try except end e/ou try finally end para proteção de recursos.

Como vimos, uma vez compilado, o código contido em um executável não pode ser alterado. Embora não possamos alterar o código podemos desviar o fluxo na pilha de processamento caso algo de errado ocorra em alguma de nossas instruções. Assim todo o código colocado após o try e antes do except será "vigiado" e caso ocorra uma exceção o fluxo é desviado e as instruções contidas entre o except e o end entram em cena. O código da **Listagem 1** mostra um exemplo de código com tratamento de exceção.

Já no caso da proteção de recurso, todo o código colocado entre o finally e o end invariavelmente será executado, assim um bloco de proteção de recurso serve para que possamos garantir que um recurso alocado será liberado independente de qualquer coisa. Observe a **Listagem 2**, ela mostra o mesmo código da **Listagem 1** porém reparem que antes de iniciarmos nossa rotina nós alteramos o cursor do mouse para uma ampulheta. Para garantir que o ponteiro do mouse volte ao seu estado normal, colocamos a instrução que altera o cursor para default dentro do bloco de proteção de recurso. Desta forma aconteça o que acontecer temos a garantia de que o curso do mouse voltará ao seu estado normal. Faça um teste, remova o bloco de proteção de recurso e force uma divisão por zero. Verá que após dar o OK na mensagem de erro o cursor do mouse permanecerá como uma ampulheta. A **Listagem 3** mostra a união do código da **Listagem 1** com a **Listagem 2**, ou seja, tratamento de exceção com proteção de recurso.

Tratando exceções

Para que possamos entender na prática como funciona o tratamento de exceções no Delphi vamos criar um pequeno exemplo e ir evoluindo, aplicando algumas técnicas que serão muito úteis no nosso dia a dia. Abra o Delphi e crie uma nova aplicação. Insira no form principal 3 Labels, 3 Edits e 1 Buttom. Disponha-os conforme a **Figura 2.** A ideia é criar uma rotina bastante simples para efetuar uma operação de divisão. No evento OnClick do botão dividir insira o código da **Listagem 4.**

Nota do DevMan

Em ciência da computação, uma pilha de chamada (ou pilha de execução) é uma pilha que armazena informações sobre as subrotinas ativas num programa de computador. Seu principal uso é registrar o ponto em que cada subrotina ativa deve retornar quando termina de executar. Ou seja, a pilha guarda o ponto onde a rotina A chamou a rotina B para que quando a rotina B termine o programa volte para o ponto exato na rotina A.

Listagem 1. Exemplo de código com tratamento de exceção

```
var
 Valor1, Valor2, Resultado: Double;
begin
 try
  Valor1 := 10;
  Valor2 := 0;
  Resultado := Valor1 / Valor2;
  ShowMessage('Resultado:
  '+FloatToStr(Resultado));
 except
  ShowMessage('Erro ao Efetuar Cálculo');
 end;
end;
```

Listagem 2. Exemplo de código com proteção de recurso

```
var
 Valor1, Valor2, Resultado: Double;
begin
 try
  Screen.Cursor := crHourGlass;
  Valor1 := 10;
  Valor2 := 0;
  Resultado := Valor1 / Valor2;
  ShowMessage('Resultado:
  '+FloatToStr(Resultado));
 finally
  Screen.Cursor := crDefault;
 end;
end;
```

Listagem 3. Exemplo de código com proteção de recurso e tratamento de exceção

```
var
 Valor1, Valor2, Resultado: Double;
begin
 try
  try
   Screen.Cursor := crHourGlass;
   Valor1 := 10;
   Valor2 := 0;
   Resultado := Valor1 / Valor2;
   ShowMessage('Resultado:'+FloatToStr(Resultado));
  except
   ShowMessage('Erro ao Efetuar Cálculo');
  end;
 finally
  Screen.Cursor := crDefault;
 end;
end;
```

Nota

Quando executar a aplicação de dentro do Delphi dê preferência a executá-la com o atalho Ctrl+Shift+F9. Isso executa a aplicação sem suporte a depuração e com isso evitamos as mensagens de exceção geradas pela IDE.

O código é bem simples, primeiro convertemos os textos dos Edits em valores numéricos. Feito isso efetuamos uma divisão comum com os valores informados e fazemos o processo inverso, desta vez convertendo o valor numérico em texto e o exibimos no Edit resultado.

Se compilarmos a aplicação não teremos qualquer mensagem de erro e nosso executável será gerado com sucesso. Se executarmos a aplicação e a utilizarmos de forma coerente, isto é, informando números e não fazendo divisão por zero nossa aplicação se mostrará eficiente. Mas não precisamos de muito esforço para notar que nosso código está falho e suscetível a erros. Faça um teste, execute a aplicação e informe uma letra em Valor 1 e clique no botão para ver o que ocorre. A **Figura 3** exibe e exceção gerada. Repare que mesmo após clicar no botão OK da mensagem de exceção a mensagem de fim da rotina não é exibida, ou seja, a exceção interrompeu o fluxo da aplicação paralisando a pilha de execução.

Temos aí dois problemas, primeiro esta é uma mensagem desagradável de se ver, o ideal seria ter uma mensagem mais amigável para o usuário, e segundo temos que garantir que o fluxo da aplicação chegue até o fim. Para isso vamos então implementar um tratamento de exceção para nossa rotina. A **Listagem 5** exibe o código com o tratamento de exceção implementado.

Repare que agora os códigos que podem gerar exceções estão dentro de um bloco de tratamento, e caso ocorra qualquer problema quando uma destas instruções for executada o processamento será desviado para as instruções contidas entre o except e o end. Assim, ocorrida a exceção um objeto descendente de exception será instanciado pelo Delphi, a pilha será desviada para o tratamento de exceção que exibirá a mensagem personalizada e destruirá o objeto da exceção permitindo assim que o fluxo do sistema continue normalmente. Faça o mesmo teste agora, execute a aplicação e informe uma letra para o Valor 1. A **Figura 4** mostra a mensagem exibida pelo código do tratamento de exceção e em seguida a mensagem de fim da rotina.

Atente para o fato de que a mensagem de fim da rotina está fora do bloco de tratamento de exceção. Isso se faz necessário, pois se ele estivesse entre o except e o end só seria exibida em caso de erro. E se a colocássemos imediatamente antes do exception ela não seria exibida em caso de exceção na rotina. Faça o teste, altere a posição da mensagem de fim de rotina para os locais citados anteriormente, execute a aplicação e observe os resultados.

Nossa rotina ainda não está 100%. Execute a aplicação e realize uma divisão por zero, a exceção será tratada, mas a mensagem de exceção não condiz com o erro que de fato

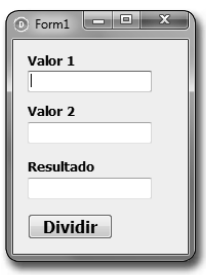


Figura 2. Form Principal

Figura 3. Exceção gerada

Figura 4. Mensagem exibida pelo tratamento de exceção

Listagem 4. Implementação do evento OnClick do botão dividir

```
var
  N1, N2, Resultado: Double;
begin
  N1 := StrToFloat(EdtValor1.Text);
  N2 := StrToFloat(EdtValor2.Text);
  Resultado := N1/N2;
  EdtResultado.Text := FloatToStr(Resultado);
  ShowMessage('Fim da Rotina');
end;
```

Listagem 5. Código com tratamento de exceção

```
var
  N1, N2, Resultado: Double;
begin
  try
    N1 := StrToFloat(EdtValor1.Text);
    N2 := StrToFloat(EdtValor2.Text);
    Resultado := N1 / N2;
    EdtResultado.Text := FloatToStr(Resultado);
  except
    MessageDlg('Erro ao tentar converter os
     valores.'+#13+
           'Verifique se os valores informados são
            válidos',
           mtInformation,[mbOK],0);
  end;
  ShowMessage('Fim da Rotina');
end;
```

Listagem 6. Tratamento específico para cada tipo de exceção

```
var
  N1, N2, Resultado: Double;
begin
  try
    N1 := StrToFloat(EdtValor1.Text);
    N2 := StrToFloat(EdtValor2.Text);
    Resultado := N1 / N2;
    EdtResultado.Text := FloatToStr(Resultado);
    ShowMessage('Fim da Rotina');
  except
    on E: EConvertError do
      MessageDlg('Erro ao tentar converter os
       valores.'+#13+
             'Verifique se os valores informados são
              válidos',
             mtInformation,[mbOK],0);
    on E: EZeroDivide do
      MessageDlg('Não é possível efetuar divisão
      por zero.',
             mtInformation,[mbOK],0);
    on E: Exception do
      MessageDlg('Um erro inesperado ocorreu.',
      mtWarning,[mbOK],0);
  end;
  ShowMessage('Fim da Rotina');
end;
```

Listagem 7. Reerguendo a exceção

```
01 var
02  N1, N2, Resultado: Double;
03 begin
04  try
05   N1 := StrToFloat(EdtValor1.Text);
06   N2 := StrToFloat(EdtValor2.Text);
07   Resultado := N1 / N2;
08   EdtResultado.Text := FloatToStr(Resultado);
09   ShowMessage('Fim da Rotina');
10  except
11  on E: EConvertError do
12   MessageDlg('Erro ao tentar converter os valores.'+#13+
13         'Verifique se os valores informados são válidos',
14         mtInformation,[mbOK],0);
15  on E: EZeroDivide do
16   MessageDlg('Não é possível efetuar divisão por zero.',
17         mtInformation,[mbOK],0);
18  on E: Exception do
19  begin
20   MessageDlg('Um erro inesperado ocorreu.', mtWarning,[mbOK],0);
21   raise Exception.Create(E.Message);
22  end;
23  end;
24  ShowMessage('Fim da Rotina');
25 end;
```

Nota do DevMan

O Windows usa handles como indexadores para se referenciar a objetos que são armazenados em uma tabela como ponteiros. Desta forma, uma aplicação Windows pode referir a uma janela ou dispositivo de contexto através de seu handle. As janelas e os dispositivos de contexto têm um tipo de handle específico. Por exemplo, um Bitmap tem um handle do tipo HBitmap. Todos os tipos de handles são do tipo THandle. Normalmente, os handles dos componentes que o possuem é conseguido através da propriedade handle.

ocorreu. Mas aí vem a questão, exceção é exceção, como saber qual erro ocorreu para que possamos exibir a mensagem correta? Como falamos, toda exceção no Delphi é um objeto do tipo Exception ou descendente deste. Assim pelo tipo da exceção podemos descobrir o que ocorreu e efetuar o tratamento de acordo com o tipo de erro. A **Listagem 6** mostra como ficou o nosso código com o tratamento específico para cada tipo de erro.

Repare que agora temos três mensagens de erro personalizadas no bloco de tratamento de exceção e para saber qual deles invocar é feito um teste no tipo da exceção gerada pelo Delphi. A sintaxe é muito simples, palavra reservada "on" seguida da variável que irá assumir o ponteiro do objeto de exceção separada de seu tipo por dois pontos (:). Essa variável não precisa ser declarada, pois faz parte da sintaxe do comando e seu nome segue a regra de nomenclatura de variáveis, sendo o E apenas uma convenção. Traduzindo o comando para o português claro seria: quando o tipo da exceção for Exxxx faça (...). Outro fato importante é que o compilador analisa as expressões de cima para baixo e encontrando uma que satisfaça a condição ele executa as instruções, destrói o objeto da exceção e segue com o fluxo normal da aplicação. Por isso é de extrema importância que se inicie o tratamento pelo tipo mais específico e se termine com o tipo mais genérico, no caso Exception.

Como comentado, a variável E assume o ponteiro do objeto que gerou a exceção e através dela temos acesso às propriedades e métodos da exceção. A principal propriedade é a message que possui a mensagem original do erro, cada exceção possui uma mensagem específica que se torna útil para o caso de reerguermos a exceção e gerarmos um log para posterior análise.

Execute a aplicação e faça o teste, informe uma letra para o valor 1, em seguida faça um divisão por zero e para finalizar informe zero para o valor 1 e para o valor 2 e assim gerar um exceção genérica.

Reerguendo a exceção e tratando em TApplication

Do ponto de vista do tratamento de exceção podemos dizer que nosso código está seguro. Porém ainda temos um caso em que a mensagem é genérica e seria interessante acompanhar a incidência deste tipo de exceção para podermos identificar o que causa este erro melhorando nossa rotina de tratamento de exceção. Mas como fazer isso se ao tratarmos a exceção o objeto é destruído? Poderíamos logo após exibir a mensagem para o usuário gerar um log de erro para posterior análise. Isso é viável no nosso exemplo que tem apenas uma rotina com tratamento de exceção, mas na prática uma aplicação pode ter dezenas, centenas de rotinas com e sem tratamento de exceção e ir alterando uma a uma não é uma tarefa nada fácil. Para casos como este podemos lançar mão do objeto Application. Quando uma exceção é gerada o fluxo é desviado e vai subindo na pilha de execução até encontrar um tratamento de exceção, não encontrando um bloco de tratamento dentro da pilha a exceção "estoura" no nível mais alto da aplicação e lá podemos implementar um tratamento.

Toda exceção não tratada nas suas rotinas internas vão parar no objeto Application antes de serem mostradas na tela para o usuário. Sendo assim podemos utilizar o evento OnException do objeto Application para interceptarmos toda e qualquer exceção não tratada na nossa aplicação. Isso é útil em dois motivos, primeiro que podemos facilmente implementar o mínimo de tratamento de exceção em uma aplicação já em produção, e segundo, nos permite reerguer exceções e a capturarmos em Application para, por exemplo, gerar um log de erros.

Para entendermos na prática como tudo isso funciona altere o código no nosso exemplo conforme a **Listagem 7**.

A alteração realizada encontra-se na linha 21, nela usamos a palavra reservada raise e criamos uma nova exceção, neste caso dizemos que estamos reerguendo a exceção. Nada nos impede de criar uma exceção de um tipo diferente, até mesmo reerguermos uma exceção de um tipo criado por nós, herdado de Exception.

Atentem para o fato de que esta nova exceção gerada não está sendo tratada e neste caso o pilha de execução será interrompida

Nota

Como estamos criando uma nova exceção que não está sendo tratada, a pilha de execução será interrompida e como não temos nenhum outro tratamento de exceção na mesma pilha o fluxo será desviado para o objeto Application. Neste caso a mensagem de fim de rotina não será exibida, pois o fluxo foi desviado para outra pilha, se é que assim podemos dizer. Com isso analise muito bem cada caso para que não corra o risco de reerguer uma exceção e impedir que instruções importantes do seu código deixem de ser executadas.

e o fluxo da aplicação será desviado para o primeiro bloco de tratamento de exceção nesta pilha. Como não há, esta exceção acabará no objeto Application e lá poderemos dar um tratamento específico para esta nova exceção. Repare que a mensagem exibida para o usuário será a mensagem amigável criada anteriormente.

Gerando log de exceções

Vamos finalizar nosso exemplo preparando nossa aplicação para interceptar exceções não tratadas e/ou reerguidas. Para isso vamos adicionar ao nosso formulário um componente ApplicationEvents. Este componente possui apenas eventos, mas eventos em nível de aplicação e um deles é o evento OnException. Vamos adicionar uma nova implementação em nosso exemplo, dê um duplo clique no evento OnException e nele adicione o código da **Listagem 8.**

Iniciamos nossa rotina criando uma variável do tipo TStringList. Ela será responsável pela geração das informações para posterior salvamento em disco. Em primeiro lugar criamos um objeto do tipo TStringList e associamos a variável Log. Em seguida utilizamos o método Exists de classe TFile para verificar se já há um arquivo de log gerado em disco. Atente aqui para dois fatos, primeiro que como não passamos o caminho completo do arquivo ele assumirá que o arquivo se encontra no mesmo diretório onde roda nossa aplicação e segundo que o nome do arquivo nada mais é que a data atual no formato yyyymmdd. Assim teremos apenas um arquivo de exceção por dia, isso se todo dia ocorrer pelo menos uma exceção.

Caso o arquivo exista nós o carregamos do disco para acrescentar a ele as informações de mais uma exceção gerada. Caso contrário, iniciamos inserindo a primeira linha do log no arquivo. Nós adicionamos apenas quatro informações ao log, a saber: a data da ocorrência do erro, a hora da ocorrência do erro, a classe da exceção gerada e a mensagem original do erro. Com estas informações no log temos como analisar a quantidade de incidência de um mesmo tipo de erro e com isso aprimoramos nosso tratamento de exceção. Para finalizar salvamos o arquivo no disco usando o mesmo padrão de nomenclatura testada no início da rotina.

Observe que nossa rotina de geração de log está entre um bloco de tratamento de exceção e um bloco de proteção de recurso. Isso se faz necessário, pois caso ocorra um erro na geração de log, nossa rotina não entrará em loop infinito. Lembre-se que qualquer exceção não tratada cai no OnException, se esta exceção está sendo gerada no próprio OnException log a rotina entraria em loop infinito. Outro detalhe é que se estamos criando um objeto do tipo TStringList, logo estamos alocando memória para isto, assim temos que garantir que o recurso alocado será liberado. Daí a necessidade do comando FreeAndNil(Log) estar dentro do bloco finally.

Para facilitar ainda mais o trabalho de análise vamos dar um print na tela no momento que a exceção for gerada, assim teremos uma ideia do que estava sendo feito no momento em que o erro ocorreu. Na seção private do formulário declare o método CapturaTela: TBitmap, e na sua implementação insira o código da **Listagem 9.**

Basicamente este método copia a área da tela para dentro do canvas do objeto TBitmap. Para que possamos fazer isso nós criamos um objeto do tipo canvas que é o objeto utilizado pelo Windows para desenhar os controles WinControl. Neste método nosso canvas é representado pela variável CV e como todo canvas necessita de um identificador, criamos a variável DC do tipo TCanvasDC, que na verdade é um alias para um HDC (Handle Device Context). Criado o canvas nós utilizamos o método CopyRect para preencher o Canvas do objeto TBitmap com a imagem da tela.

Para finalizar altere o código do evento OnExeption conforme o código da **Listagem 10**. Repare que incluímos uma nova variável, chamada ImageName, ela receberá o nome que será dado ao arquivo bmp que conterá o print da tela. O nome nada mais é do que a data e hora do momento da exceção. Além disso, incluímos mais três linhas de código, uma para gerar o nome do arquivo, a segunda que invoca o método CapturaTela e no mesmo comando salva o arquivo gerado no disco, e a terceira linha que inclui mais uma informação do log de erro, ou seja, o nome do arquivo com a imagem do erro gerado no momento da exceção.

Listagem 8. Gerando log das exceções

```
var
 Log: TStringList;
begin
 try
  Log := TStringList.Create;
  try
   if TFile.Exists(FormatDateTime('yyyymmdd', Now) + '.log') then
    Log.LoadFromFile(DateToStr(Now) + '.log');
   Log.Add('Data: ' + DateToStr(Now));
   Log.Add('Hora: ' + TimeToStr(Now));
   Log.Add('Classe do Erro: ' + E.ClassName);
   Log.Add('Mesagem de erro: ' + E.Message);
   Log.Add('');
   Log.SaveToFile(FormatDateTime('yyyymmdd', Now) + '.log');
  except
   MessageDlg('Erro ao gerar log', mtInformation,
   [mbOK], 0);
  end;
 finally
  FreeAndNil(Log);
 end;
end;
```

Listagem 9. Método CapturaTela

```
var
 DC: TCanvasDC;
 CV: TCanvas;
begin
 Result      := TBitmap.Create;
 Result.Width  := Screen.Width;
 Result.Height := Screen.Height;
 DC := GetDC(0);
 CV := TCanvas.Create;
 CV.Handle := DC;
 Result.Canvas.CopyRect
 (Rect(0,0,Screen.Width,Screen.Height),cv,Rect(0,0,Screen.Width,
  Screen.Height));
 FreeAndNil(cv);
 ReleaseDC(0,DC);
end;
```

Conclusão

O exemplo desenvolvido neste artigo pode ser aprimorado. A rotina pode ser melhorada de várias formas, além do log

Listagem 10. Rotina final de geração do log

```
var
  Log: TStringList;
  ImageName: String;
begin
  try
    Log := TStringList.Create;
    try
      if TFile.Exists(FormatDateTime('yyyymmdd', Now) + '.log') then
        Log.LoadFromFile(FormatDateTime('yyyymmdd', Now) + '.log');
        Log.Add('Data: ' + DateToStr(Now));
        Log.Add('Hora: ' + TimeToStr(Now));
        Log.Add('Classe do Erro: ' + E.ClassName);
        Log.Add('Mesagem de erro: ' + E.Message);
        ImageName := FormatDateTime('yyyymmddhhnnss', Now) + '.bmp';
        CapturaTela.SaveToFile(ImageName);
        Log.Add('ScreenShot: '+ImageName);
        Log.Add('');
        Log.SaveToFile(FormatDateTime('yyyymmdd', Now) + '.log');
    except
      MessageDlg('Erro ao gerar log', mtInformation, [mbOK], 0);
    end;
  finally
    FreeAndNil(Log);
  end;
end;
```

poderíamos disparar um email para nos notificar da ocorrência do erro. Com o tratamento de exceção no nível da aplicação podemos ainda criar nossas próprias caixas de diálogo, personalizadas para tornar as mensagens mais amigáveis. Ou ainda criarmos um pequeno framework que intercepte as mensagens de erro em inglês e converta-as para algo mais familiar para o usuário e irmos com o passar do tempo aprimorando esse framework para que contemple cada vez mais mensagens.

As possibilidades são infinitas, o importante é garantir o desempenho e a segurança de suas aplicações no que diz respeito à tolerância a falhas. Lembre-se, nenhum programa está livre de erros, porém devemos estar preparados para lidar com eles.

Rodrigo Carreiro Mourão
rodrigocarreiro@rmfactory.com.br
Analista, programador, consultor, instrutor e editor técnico da revista Clube Delphi. Instrutor certificado pela Borland, possui ainda o título de Delphi Product Certified. É sócio da RM Factory Soluções em TI e da Szeryng Traduções e Informática. Atua como analista de TI do Grupo Editorial Nacional e é instrutor de treinamentos oficiais Embarcadero no Rio de Janeiro. Como palestrante atuou na última Borcon entre outros eventos de Delphi no Brasil. Membro ativo na comunidade e apaixonado pelo Delphi.

Compiladores Freeware
http://www.thefreecountry.com/compilers/index.shtml

Lista de Compiladores
http://en.wikipedia.org/wiki/List_of_compilers

Blogs do autor
http://www.rodrigocarreiro.com.br
http://twitter.com/carreiromourao

Dê seu feedback sobre esta edição!

A Clubedelphi tem que ser feita ao seu gosto. Para isso, precisamos saber o que você, leitor, acha da revista!

Dê seu voto sobre este artigo, através do link:
www.devmedia.com.br/clubedelphi/feedback

Cursos Online

WebMobile PLUS

A **Revista WebMobile** oferece para seus assinantes uma série de Cursos Online de alto padrão de qualidade **.Conheça abaixo os cursos já disponíveis.**

Curso de .net em destaque

Aplicação completa de orçamento Doméstico no Visual Studio 2005

Confira neste curso online como criar uma aplicação completa no Visual Studio 2005, usando ASP.NET, Web Services, Mobile e muito mais! Veja como criar uma aplicação de orçamento doméstico, usando diagrama de classes de uma forma muita produtiva, criar classes de negócios de acesso a dados.

Confira o plano de aula completo:
www.devmedia.com.br/domesticovs

Curso de Java em destaque

Introdução ao uso de Web Services em Java ME

Confira neste curso online como realizar o processo completo de desenvolvimento de um servidor de WebService utilizando o Netbeans, com servidor de aplicação GlassFish e o acesso ao mesmo utilizando aplicativo celular (JME) com a JSR-172.

Confira o plano de aula completo:
www.devmedia.com.br/webservicesjme

Assine a **WebMobile** e comece já seu treinamento!
www.devmedia.com.br/assine

A sua melhor opção de aprendizagem!

Em breve mais novidades para você! www.devmedia.com.br/curso

DevMedia **Mais Informações: www.devmedia.com.br/central - Tel.: 21 3382-5038**

Hospede seus sites de forma simples, rápida e ilimitada.

Plano **Básico**

- 1 domínio principal
- 50 domínios hospedados
- Transferência **ilimitada**
- Espaço **ilimitado**
- 10 contas de e-mail
- PHP, ASP e ASP.NET
- Painel de controle Plesk
- Suporte 24hs

Tudo isso por apenas

R$ **15,90** por mês
pagamento trimestral

Assine agora mesmo e descubra por que Hospedix é a hospedagem de sites preferida entre os profissionais.

- Conheça também nossos planos profissionais com registro de domínio grátis e isenção de taxas anuais enquanto for cliente.

HOSPEDIX
HOSPEDAGEM PROFISSIONAL
www.hospedix.com.br

30 DIAS GRÁTIS

Ao assinar, digite o código de desconto exclusivo para leitores desta revista e ganhe 30 dias inteiramente grátis.

RVSTMEDIA

As promoções anunciadas não são cumulativas com outras promoções e são válidas apenas para clientes Hospedix ativos.